IMPLEMENTASI ALGORITME TRIVIUM UNTUK MENGAMANKAN KOMUNIKASI DATA MASTER-SLAVE PADA PERANGKAT BERBASIS MODUL KOMUNIKASI NRF24L01

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Sara Yosephina
NIM: 145150200111095

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

IMPLEMENTASI ALGORITME TRIVIUM UNTUK MENGAMANKAN KOMUNIKASI DATA MASTER-SLAVE PADA PERANGKAT BERBASIS MODUL KOMUNIKASI NRF24L01

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Sara Yosephina
NIM: 145150200111095

Skripsi ini telah diuji dan dinyatakan lulus pada
03 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Adhitya Bhawiyuga, S.Kom, M.Sc	Ari Kusyanti, S.T, M.Sc
NIK: 2014058907201001	NIK: 2011028312282001

Mengetahui
Ketua Jurusan Informatika

Tri Astoto Kurniawan , S.T, M.T, Ph.D
NIP: 197105182003121001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 19 Juli 2018

Sara Yosephina

NIM: 145150200111095

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa, karena atas limpahan rahmat dan cinta kasih-Nya sehingga penulis dapat menyelesaikan tugas akhir skripsi yang berjudul "Implementasi Algoritme Trivium Untuk Mengamankan Komunikasi Data *Master-Slave* Pada Perangkat Berbasis Modul Komunikasi Nrf24l01". Atas bantuan moral dan materil yang diberikan kepada penulis, maka penulis juga mengucapkan banyak terima kasih kepada:

1. Ibunda Ramsia Tambunan yang selalu mendoakan serta memberikan dukungan untuk tetap semangat dalam proses menyelesaikan tugas akhir skripsi.
2. Ayahnda Togarma Marbun yang memberikan doa dan dukungan sehingga penulis dapat dengan lancar menyelesaikan skripsi.
3. Saudara-saudara penulis: Allan Marbun, Alfred Marbun dan Anatoli marbun yang tetap mendukung dan memberikan semangat kepada penulis.
4. Bapak Adhitya Bhawiyuga, S.Kom, M.Sc. selaku pembimbing satu yang telah membimbing serta memberikan ilmu, saran dan motivasi kepada penulis.
5. Ibu Ari Kusyanti, S.T, M.Sc. selaku pembimbing dua yang telah meminjamkan Arduino Mega 2560, membimbing serta memberikan ilmu, saran dan motivasi kepada penulis.
6. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya.
7. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D. selaku Ketua Jurusan Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya.
8. Seluruh civitas akademika Fakultas Ilmu Komputer Universitas Brawijaya yang telah banyak memberi bantuan dan dukungan selama penulis menempuh studi Teknik informatika di Universitas Brawijaya dan selama penyelesaian skripsi.
9. Leo Silalahi dan Tania Oka Sianturi yang memberikan dukungan, doa dan semangat serta menemani penulis selama proses penyelesaian skripsi ini.
10. Teman-teman dekat penulis: Silvia Aprilla, Tryse Biantong, Josua Fernando, Inggrid Melanika, Dinda Agnes, Olivia Sitanggang, dan yang selalu mendukung, memberi doa, semangat dan bantuan serta menemani penulis selama proses pengerjaan skripsi.
11. Teman kontrakan dan kost penulis: Juni Simangunsong, Debora Napitupulu, Hotma Yani, Inggrid Siahaan dan Ferina Gurning yang telah menemani penulis selama proses menempuh studi di kota Malang.
12. Keluarga Besar PMK Daniel yang telah mendoakan dan mendukung penulis selama menempuh studi di Fakultas Ilmu computer dan dalam proses penyelesaian skripsi.

13. Semua teman-teman seperjuangan skripsi Teknik Informatika yang telah mendukung, memberi doa, semangat dan bantuan serta menemani penulis selama proses pengerjaan skripsi ini.

Penulis menyadari bahwa dalam penyusunan tugas akhir skripsi ini masih terdapat banyak kekurangan, sehingga kritik dan saran yang membangun sangat penulis harapkan. Akhir kata penulis berharap skripsi ini dapat bermanfaat bagi semua pihak yang menggunakannya.

Malang, 19 Juli 2018

Penulis

saraym@student.ub.ac.id

ABSTRAK

Internet of Things (IoT) merupakan salah satu konsep jaringan yang menghubungkan berbagai berbagai perangkat sehingga memiliki kemampuan untuk berkomunikasi dan saling bertukar informasi. Perangkat-perangkat tersebut dapat menggunakan maupun menghasilkan layanan-layanan dan saling berkerjasama untuk mencapai tujuan bersama. Namun salah satu yang menjadi isu kelemahan dari pengimplementasian IoT yaitu masalah keamanan data dan privasi. NIST (*National Institute of Standard Technology*) merupakan institusi yang menaungi standar keamanan data. NIST menyelenggarakan projek eSTREAM untuk mencari algoritme *stream cipher* baru untuk keamanan *software* maupun *hardware*. Algoritme Trivium merupakan salah satu kandidat yang terpilih untuk keamanan dari sisi *hardware*. Algoritme Trivium lebih unggul dibandingkan dengan algoritme yang lain karena arsitekturnya memiliki tingkat kompleksitas yang rumit dan tidak membutuhkan sumber daya yang besar sehingga cocok untuk digunakan pada perangkat yang memiliki daya yang rendah. Pada penelitian ini menjelaskan bagaimana memanfaatkan algoritme Trivium untuk mengamankan komunikasi data *master-slave* pada perangkat berbasis modul komunikasi NRF24L01. Penggunaan NRF24L01 pada penelitian ini dikarenakan modul tersebut memiliki harga yang tejangkau dan didesain untuk layanan yang membutuhkan daya yang sangat rendah. Terdapat 5 pengujian yang dilakukan diantaranya validasi *test vector*, analisis performansi melalui waktu proses enkripsi dan dekripsi, analisis performansi waktu pengiriman dari *master* ke *slave* menggunakan modul komunikasi NRF24L01, dan uji keamanan dengan melakukan *sniffing*. Hasil dari penelitian ini menunjukkan bahwa validasi *keystream* telah sesuai dengan *test vector*. Hasil performasi waktu pemrosesan enkripsi memiliki rata-rata 695,8 ms untuk ukuran data 8 bit, 951,2 ms untuk ukuran data 16 bit dan 1460,2 ms untuk ukuran data 32 bit. Dan Hasil performasi waktu pemrosesan dekripsi memiliki rata-rata 443,5 ms untuk ukuran data 8 bit, 447,8 ms untuk ukuran data 16 bit dan 448,4 ms untuk ukuran data 32 bit. Dan hasil dari pengujian keamanan dengan melakukan *sniffing* menunjukkan bahwa algoritme Trivium dapat melakukan enkripsi data dengan baik sehingga proses *sniffing* hanya dapat menyadap data yang telah dienkripsi.

Kata kunci: *Internet of Things*, Keamanan, Algoritme Trivium, *master-slave*, NRF24L01

ABSTRACT

Internet of Things(IoT) refers to the network of identifiable and addressable objects that have ability to communicate and exchange information regarding themselves. Objects in IoT can use or produce services and work together to attain a common goal. IoT has to deal with security and privacy issue that may slowing down its widespread implementation. NIST (National Institute of Standard Technology) is an institution that deal with data security standards. NIST organized the STREAM project to look for new stream cipher algorithms for both software and hardware security. Trivium algorithm is one of the candidates selected for security from the side of the hardware. Trivium algorithm is superior from other algorithms because its architecture has a complex level of complexity and does not require high resources so that make suitable for devices with low power. This research explains how to use Trivium algorithm to make secure master-slave data communications on NRF24L01 communication-based device. Researchers chose NRF24L01 due to the use a module that has affordable prices and designed for services that require very low power. There are 5 tests performed including validation test vector, performance analysis through the validation time keystream and data encryption ecryption, performance analysis of delivery time from master to slave using communication module NRF24L01, and testing with sniffing. Results of this resesarch indicate that the keystream validation in accordance with test vector. The performance of encryption processing time has an average of 695.8 ms for 8 bit data size, 951.2 ms for 16 bit data size and 1460.2 ms for 32 bit data size. And the results of the decryption processing time performance have an average of 443.5 ms for 8 bit data size, 447.8 ms for 16 bit data size and 448.4 ms for 32 bit data sizeAnd the results of the test by sniffing shows that the Trivium algorithm can perform data encryption well with the sniffing process can only sniff the data that has been encrypted.

Keywords: Internet of Things, Security, Trivium Algorithm, master-slave, NRF24L01

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Internet of Things (IoT) merupakan salah satu konsep jaringan yang menghubungkan berbagai perangkat sehingga memiliki kemampuan untuk berkomunikasi dan saling bertukar informasi serta perangkat-perangkat yang ada dalam IoT tersebut dapat menggunakan maupun menghasilkan layanan-layanan dan saling bekerjasama untuk mencapai tujuan bersama (Ernita, 2015). Komunikasi antar perangkat-perangkat tersebut dapat terjadi dengan adanya modul *tranceiver* yang digunakan sebagai media transmisi nirkabel.

Salah satu modul *transceiver* yang memanfaatkan *radio frequency* dalam transmisi data digital yaitu NRF24L01. NRF24L01 merupakan modul komunikasi jarak jauh yang menggunakan frekuensi pita gelombang radio 2.4-2.5 GHz ISM (*Industrial Scientific and Medical*). NRF24L01 memiliki kecepatan sampai 2Mbps dengan pilihan opsi *date rate* 250 Kbps, 1 Mbps, dan 2 Mbps. *Transceiver* terdiri dari synthesizer frekuensi terintegrasi, kekuatan amplifier, osilator kristal, demodulator, modulator dan Enhanced ShockBurst ™ mesin protokol, output daya, saluran frekuensi, dan setup protokol yang mudah diprogram melalui antarmuka SPI (*Serial Periperal interface*). Modul ini didesain untuk jaringan nirkabel yang membutuhkan daya sangat rendah (Nordic Semiconductor ASA., 2007). Namun, yang menjadi salah satu kelemahan dari penggunaan modul NRF24L01 yaitu rentannya terhadap serangan keamanan data. Hal ini terjadi karena belum adanya mekanisme yang menjamin keamanan transmisi pertukaran data yang terjadi antar modul NRF24L01. Oleh karena itu, diperlukan adanya mekanisme yang dapat memastikan komunikasi data yang dilakukan dapat berjalan dengan aman.

Kriptografi adalah salah satu mekanisme yang dapat digunakan untuk memastikan keamanan dari suatu data. Kriptografi menyediakan teknik, mekanisme dan alat yang digunakan untuk mengamankan komunikasi dan otentikasi pribadi pada internet dan jaringan terbuka lainnya (Nesrine, 2017). Dalam kriptografi proses penyandian *plaintext* menjadi *ciphertex*t disebut enkripsi (*encryption*) atau enciphering (standard nama menurut ISO 7498-2). Sedangkan proses mengembalikan *ciphertext* menjadi *plaintext* semula dinamakan dekripsi (*decryption*) atau deciphering (standard nama menurut ISO 7498-2).

Algoritme Trivium merupakan algoritme yang dihasikan dari *European project* eSTREAM yang diselenggarakan oleh *National Institute of Standard and Tecnology* (NIST) dimana eStream project bertujuan untuk memperbarui atau mengganti algoritme *steam cipher* yang telah lama dengan yang baru. Algoritme Trivium memiliki *internal state* yang berukuran 288 bit dan kunci dengan panjang 80 bit. Salah satu hal yang menjadi alasan penggunaan algoritme Trivium pada penelitian ini yaitu tidak adanya hak paten atas algoritme ini sehingga bebas

digunakan oleh siapa saja dan telah ditetapkan sebagai standar international oleh ISO / IEC 29192-3.

Penelitian sebelumnya yang dilakukan oleh Paris Kitsos dengan judul penelitian "*FPGA-based performance analysis of stream ciphers ZUC, Snow3g, Grain V1, Mickey V2, Trivium and E0*". Penelitian ini bertujuan untuk menganalisis kinerja dari keenam *stream cipher* tersebut. ZUC, Snow3g dan E0 merupakan *stream cipher* yang telah sering digunakan untuk keamanan nirkabel. Sedangkan Grain V1, Mickey V2 dan Trivium merupakan *stream cipher* yang didesain untuk implementasi *hardware* oleh eSTREAM *Project* yang diselenggarakan oleh NIST. Enam *stream cipher* ini desainnya diimplementasikan menggunakan bahasa VHDL sedangkan untuk implementasi perangkat keras menggunakan perangkat FPGA. Pada penelitian tersebut menjelaskan perbandingan dari enam stream cipher dalam hal *performance,* konsumsi daya dan *throughput*. Throughput tertinggi dicapai oleh algoritme Snow3g namun menggunakan konsumsi daya yang tinggi. Hal yang sama juga terjadi pada algoritme ZUC yang mencapai throughput yang tinggi namun mengkonsumsi daya yang tinggi pula. Dan hasil yang dicapai algoritme Trivium yaitu kebutuhan daya rendah namun tetap memiliki throughput yang tinggi. Sedangkan algoritme Grain V1 dan E0 memberikan hasil terendah dalam hal *throughput*. Dan algoritme Mickey V2 memberikan hasil terendah pada konsumsi daya namun memiliki *throughput* yang rendah pula.

Pada penelitian ini, penulis menerapkan mekanisme pengamanan komunikasi data *master-slave* pada perangkat berbasis modul komunikasi NRF24L01 dengan mengimplementasikan algoritme Trivium. Dengan penelitian ini diharapkan menjadi langkah awal untuk menciptakan mekanisme keamanan data yang berkonsep IoT yang membutuhkan daya rendah. Dalam hal ini, terdapat satu perangkat yang berperan sebagai master dan satu perangkat yang lain berperan sebagai slave. Perangkat yang berperan sebagai slave akan melakukan proses enkripsi dengan menggunakan keystream yang telah diproses sebelumnya. Selanjutnya hasil enkripsi tersebut akan dikirimkan ke master menggunakan modul NRF24L01 untuk didekripsi dengan menggunakan keystream yang telah diproses pula.

1.2 Rumusan Masalah

Berikut rumusan masalah dari penelitian ini:

1. Bagaimana mekanisme implementasi algoritme Trivium untuk mengamankan komunikasi data *master-slave* pada perangkat berbasis modul komunikasi NRF24L01?
2. Bagaimana validasi *keystream* algoritme Trivium untuk mengamankan komunikasi data *master-slave* pada perangkat berbasis modul komunikasi NRF24L01?
3. Bagaimana kinerja enkripsi dengan menggunakan modul komunikasi NRF24L01?

4. Bagaimana kinerja pemrosesan enkripsi dan dekripsi algoritme Trivium untuk mengamankan komunikasi data *master-slave* pada perangkat berbasis modul komunikasi NRF24L01?

1.3 Tujuan

Berikut tujuan dari penelitian ini:

1. Mengimplementasikan algoritme Trivium untuk mengamankan komunikasi data *master-slave* pada perangkat berbasis modul komunikasi NRF24L01.
2. Mampu melakukan validasi *dari keystream* algoritme Trivium untuk mengamankan komunikasi data *master-slave* pada perangkat berbasis modul komunikasi NRF24L01.
3. Menganalisis kinerja enkripsi dengan menggunakan modul komunikasi NRF24L01.
4. Menganalisis kinerja pemrosesan enkripsi dan dekripsi algoritme Trivium untuk mengamankan komunikasi data *master-slave* pada perangkat berbasis modul komunikasi NRF24L01.

1.4 Manfaat

Berikut manfaat dari penelitian ini:

1. Mampu memberikan solusi terhadap keamanan komunikasi data menggunakan algoritme Trivium pada bidang yang berkonsep IoT.
2. Memberikan wawasan pengetahuan yang baru kepada pembaca mengenai algoritme keamanan komunikasi data yang dapat digunakan pada modul komunikasi.

1.5 Batasan Masalah

Berikut merupakan batasan masalah dari penelitian ini:

1. Kunci dan *Initial Value* (IV) merupakan bilangan hexadisimal.
2. Masukan *plaintext* merupakan bilangan hexadesimal.
3. Implementasi sistem ini hanya akan mengenkripsi paket data 8,16 dan 32 bit.

1.6 Sistematika Pembahasan

Sistematika penulisan pada penelitian ini sebagai berikut:

BAB I : Pendahuluan

Bab ini berisikan mengenai penjelasan latar belakang yang menjadi alasan untuk penulis menyusun topik penelitian, rumusan masalah, tujuan, manfaat, batasan masalah, dan sistematika pembahasan mengenai implementasi

algoritme Trivium untuk mengamankan komunikasi data *master-slave* pada modul komunikasi NRF24L01.

BAB II : Landasan Kepustakaan

Bab ini menjelaskan mengenai teori-teori dan referensi serta materi yang mendukung mengenai algoritme Trivium, modul komunikasi NRF24L01 dan *microcontroller* arduino yang digunakan dalam membantu penulisan dan menemukan jawaban atas rumusan masalah.

BAB III : Metodologi

Bab ini menjelaskan tentang metode dan langkah kerja yang dilakukan dalam penulisan tugas akhir yang terdiri dari studi literatur, rekayasa kebutuhan system dan perancangan implementasi dari algoritme Trivium pada modul komunikasi NRF24L01.

BAB IV : Rekayasa Kebutuhan

Bab ini berisi mengenai kebutuhan perangkat lunak maupun perangkat keras yang menunjang dalam perancangan dan implementasi algoritme Trivium untuk mengamankan komunikasi data *master-slave* pada modul komunikasi NRF24L01.

BAB V :Perancangan dan Implementasi

Bab ini berisi tentang perancangan dan implementasi dari algoritme Trivium pada modul komunikasi NRF24L01.

BAB VI : Pengujian dan Analisis

Bab ini menjelaskan hasil dari pengujian dan analisis dari performansi dari algoritme Trivium pada modul komunikasi NRF24L01.

BAB VII : Penutup

Bab ini berisi ringkasan kesimpulan dari pencapaian dari pengimplementasian algoritme Trivium pada modul komunikasi NRF24L01 dan saran untuk pengembangan lebih lanjut.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab landasan kepustakaan ini, akan dikaji mengenai teori-teori, referensi dan beberapa penelitian yang telah dilakukan sebelumnya terkait dengan penelitian yang akan dilaksanakan. Penelitian ini menggunakan tiga penelitian sebelumnya yang dijadikan sebagai kajian pustaka. Kajian pustaka yang pertama yaitu penelitian yang dilakukan oleh Christophe De Canniere dan Bart Preneel yang merupakan penelitian awal dari terciptanya Algoritme Trivium dengan judul penelitian "*A Stream Cipher Construction Inspired by Block Cipher Design Principle"*. Kajian pustaka yang kedua yaitu penelitian yang dilakukan oleh Paris Kitsos yaitu "*FPGA-based performance analysis of stream ciphers ZUC, Snow3g, Grain V1, Mickey V2, Trivium and E0*" yang bertujuan untuk menganalisis performa dari keenam *stream cipher* tersebut. Penelitian selanjutnya yang dijadikan sebagai kajian pustaka yaitu penelitian yang dilakukan oleh Ken Cai dan Liang Xiaoying mengenai implementasi Algoritme kriptografi AES pada modul komunikasi NRF24L01 dengan judul penelitian "*A SOPC-BASED Evaluation of AES for 2.4 GHz Wireless Network*".

2.1 Kajian Pustaka

Pada bagian sub bab ini, akan dijelaskan mengenai perbedaan dari penelitian yang telah dilakukan sebelumnya dengan penelitian yang akan dilakukan.

Tabel 2.1 Kajian Pustaka

No.	Nama Peneliti, Judul Penelitian dan Tahun Penelitian	Persamaan Penelitian	Perbedaan Penelitian	
			Penelitian sebelumnya	Penelitian yang akan dilakukan
1.	Christophe De Canniere and Bart Preneel, *Trivium - A Stream Cipher Construction Inspired by Block Cipher Design Principle*, 2006	Menggunakan Algoritme Trivium	-	-
2.	Paris Kitsos, *FPGA-based performance analysis of stream ciphers ZUC, Snow3g, Grain V1, Mickey V2, Trivium and E0*, 2013	Menggunakan Algoritme Trivium	Analisis performansi Algoritme Trivium pada FPGA	Implementasi Algoritme Trivium pada modul komunikasi NRF24L01
3	Ken, C. and Xiaoying, L., *A SOPC-BASED Evaluation of AES for 2.4 GHz Wireless Network,* 2012	Menggunakan modul komunikasi NRF24L01	Implementasi Algoritme AES pada modul komunikasi NRF24L01	Implementasi Algoritme Trivium pada modul komunikasi NRF24L01

Tabel 2.1 merupakan kajian pustaka yang berisi perbandingan mengenai penelitian sebelumnya dengan penelitian yang akan dilakukan dan merupakan acuan dalam merancang penelitian implementasi algoritme Trivium untuk mengamankan komunikasi data master-slave berbasis modul komunikasi NRF24L01.

2.2 Dasar Teori

Pada sub bab ini akan dijelaskan mengenai teori kriptografi, algoritme Trivium, modul komunikasi NRF24L01 serta microcontroller arduino.

2.2.1 Kriptografi

Kriptografi adalah ilmu yang berdasarkan pada teknik matematika yang erat kaitannya dengan keamanan informasi seperti kerahasiaan, keutuhan data dan otentikasi dengan kata lain kriptografi bukan hanya berperan sebagai penyembunyian pesan melainkan lebih mengarah kepada sekumpulan teknik yang menyediakan keamanan informasi (Arinten, 2017). *Confidentiality, integrity*, dan *availability* merupakan aspek penting dari keamanan informasi. *Confidentiality* mengacu kepada kerahasiaan informasi dimana informasi tidak dapat dilihat maupun diakses oleh orang yang tidak berhak untuk mendapatkan informasi tersebut. *Integrity* mengarah kepada keaslian dan kepercayaan informasi bahwa informasi tersebut merupakan informasi yang asli yang tidak diubah oleh orang yang tidak berhak. Sedangkan *Availability* merupakan ketersediaan informasi agar dapat diakses kapanpun dan dimanapun oleh orang memiliki hak akses.

Pada dasarnya kriptografi terdiri dari beberapa komponen yaitu:

1. Enkripsi: merupakan proses pengamanan data yang dikirimkan sehingga terjaga kerahasiaannya dengan mengubah pesan awal menjadi bentuk pesan yang tidak dapat dimengerti atau *ciphertext*.
2. Dekripsi: merupakan kebalikan dari dekripsi yaitu proses mengembalikan pesan yang telah dienkripsi ke bentuk pesan awal atau *plaintext.*
3. *Ciphertext*: merupakan suatu pesan yang telah melalui proses enkripsi dan pesan yang ada telah menjadi bentuk pesan yang tidak dimengerti karena karakter-karakter yang tidak mempunyai makna atau arti.
4. *Plaintext*: sering juga disebut sebagai *cleartext* yang merupakan pesan asli dan memiliki makna.

2.2.2 Confidentiality

Confidentiality merupakan aspek yang menjamin kerahasiaan dari data maupun informasi. Istilah *confidentiality* dapat mencakup dua konsep terkait yaitu *data confidentiality* dan *privacy. Data confidentiality* bertujuan untuk memastikan bahwa data maupun informasi tidak dapat diakses oleh orang yang tidak berhak. Sedangkan *privacy* untuk memastikan pembatasan akses terhadap pihak yang memiliki hak hanya dapat mengakses. Secara umum dapat dikatakan

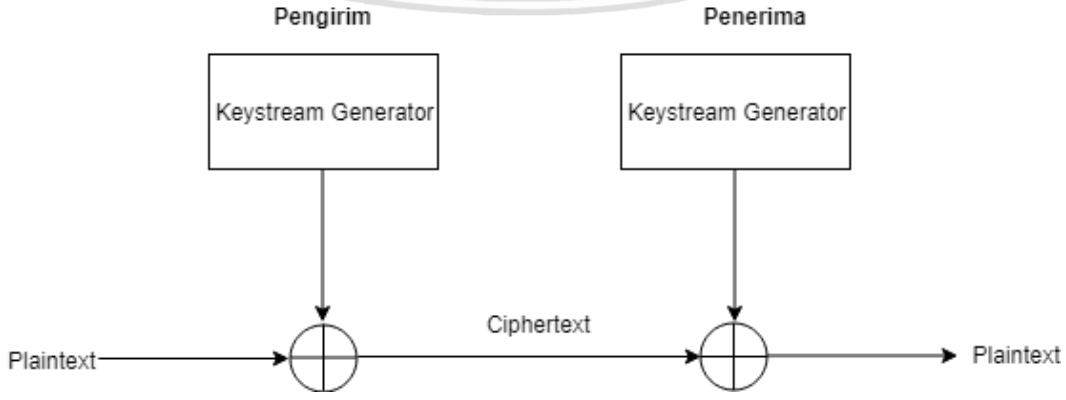
bahwa *confidentiality* mengandung makna bahwa informasi yang tepat terakses orang pihak yang berhak pula. Salah satu cara yang dapat dilakukan untuk menjamin *confidentiality* yaitu dengan menggunakan teknik kriptografi dengan melakukan metode enkripsi dan dekripsi (Chalifa, 2015).

2.2.3 Kriptografi Simetris

Sistem kriptografi simetris (*symmetric Cyptosystem*), sering juga disebut sebagai algoritma konvensional merupakan algoritma dimana enkripsi dapat dihitung dari kunci dekripsi dan sebaliknya, artinya kunci yang digunakan untuk enkripsi sama dengan kunci yang digunakan untuk dekripsi (Riyadi, 2017). Algoritma ini mengharuskan pengirim dan penerima untuk menyepakati kunci yang akan digunakan sebelum mereka dapat berkomunikasi dengan aman. Keamanan algoritma simetris terletak pada kunci, kebocoran kunci berarti siapapun bisa mengenkripsi dan mendekripsi pesan. Algoritma simetris dapat dibagi menjadi dua kategori, yang pertama beroperasi pada *plaintext* per satu bit atau biasa disebut dengan *stream cipher* dan yang beroperasi pada *plaintext* dalam bentuk per kelompok bit. Kelompok-kelompok bit yang disebut dengan blok dan biasa disebut dengan *block cipher*.

2.2.4 Stream Cipher

Stream cipher merupakan algoritma kriptografi yang beroperasi pada *plaintext* dan *ciphertext* dalam bentuk bit tunggal dimana rangkaian bit dienkripsi maupun didekripsi bit per bit. Pada *stream cipher*, bit hanya mempunyai dua nilai, sehingga proses enkripsi hanya menyebabkan bua keadaan pada bit tersebut yaitu berubah atau tidak berubah. Dua keadaan tersebut ditentukan oleh kunci enkripsi yang digunakan atau biasa disebut dengan *keystream*. *Keystream* dihasilkan dari sebuah pembangkit-aliran kunci dan disebut dengan *keystream generator*. *Keystream* di-XOR-kan dengan aliran bit-bit *plaintext* untuk menghasilkan aliran bit *ciphertext*. Pada Gambar 2.1 menjelaskan mengenai proses *stream cipher*:



Gambar 2.1 *Stream Cipher*

Gambar 2.1 merupakan penjelasan dari konsep *stream cipher*. Bit-bit kunci yang digunakan untuk proses enkripsi dan dekripsi disebut dengan *keystream* dan *keystream* di bangkitkan oleh *keystream generator*. Untuk menghasilkan *ciphertext*, *plaintext* harus di-XOR-kan dengan *keystream*. Sedangkan untuk mengembalikan ke *plaintext*, *ciphertext* harus di-XOR-kan dengan *keystream*.

2.2.5 Algoritme Trivium

Algoritme Trivium adalah *synchronous stream cipher* dan salah satu kandidat *stream cipher* yang didesain dengan diorientasikan pada implementasi *hardware* dari *European project* eSTREAM yang diselenggarakan oleh *National Institute of Standard and Tecnology* (NIST). Algoritme Trivium didesain untuk dapat menghasilkan *keystream* hingga 2^{64} bit, dan memiliki *state internal* yang berukuran 288 bit serta kunci maupun *initial value* (IV) dengan panjang 80 bit. Proses menghasilkan *keystream* pada algoritme Trivium terdiri dari dua fase. Yang pertama *keystream generator* atau inisialisasi *internal state* sejumlah 288 bit yang dinotasikan dengan (S_1,S_2 , S_3 , ... ,S_{288}) menggunakan IV dan kunci 80 bit. Proses pembangkitan *keystream* mengandung proses perulangan yang mengekstrak nilai dari 15 bit bit spesifik *state* dan menggunakannya untuk mengupdate 3 bit dari *state* dan menghitung 1 bit dari *keystream* Z_i. Bit-bit dari *state* kemudian dirotasikan dan proses berulang hingga $N \leq 2^{64}$ *keystream* yang butuhkan.

For i = 1 to N do

$t_1 \leftarrow S_{66} + S_{93}$

$t_2 \leftarrow S_{162} + S_{177}$

$t_3 \leftarrow S_{243} + S_{288}$

$Z_i \leftarrow t_1 + t_2 + t_3$

$t_1 \leftarrow t_1 + S_{91} . S_{92} + S_{171}$

$t_2 \leftarrow t_2 + S_{175} . S_{176} + S_{264}$

$t_3 \leftarrow t_3 + S_{286} . S_{287} + S_{69}$

$(S_1, S_2, \ldots , S_{93}) \leftarrow (t_3, S_1, \ldots, S_{92})$

$(S_{94}, S_{95}, \ldots , S_{177}) \leftarrow (t_1, S_{94}, \ldots, S_{176})$

$(S_{178}, S_{279}, \ldots , S_{288}) \leftarrow (t_2, S_{178}, \ldots, S_{287})$

Selanjutnya perubahan kondisi *state internal* untuk menghasilkan *keystream* yang dilakukan dengan memasukkan 80 bit kunci dan 80 bit IV kedalam *state internal* yang berukuran 288 bit, dan mengisi sisa bit *state internal* dengan bilangan biner 0, kecuali untuk S_{286}, S_{287}, S_{288}. Kemudian *state* dirotasikan sebanyak 1152 kali dengan cara yang dijelaskan seperti diatas tanpa proses pembangkitan nilai *keystream.*

$(S_1, S_2, \ldots , S_{93}) \leftarrow (K_1, \ldots, K_{80}, 0, \ldots, 0)$

$(S_{94}, S_{95}, \ldots , S_{177}) \leftarrow (IV_1, \ldots, IV_{80}, 0, \ldots,0)$

$(S_{94}, S_{95}, \ldots, S_{177}) \leftarrow (0, \ldots, 1, 1, 1)$

For i = 1 to 4.288 do

$t_1 \leftarrow S_{66} + S_{91} . S_{92} + S_{93} + S_{171}$

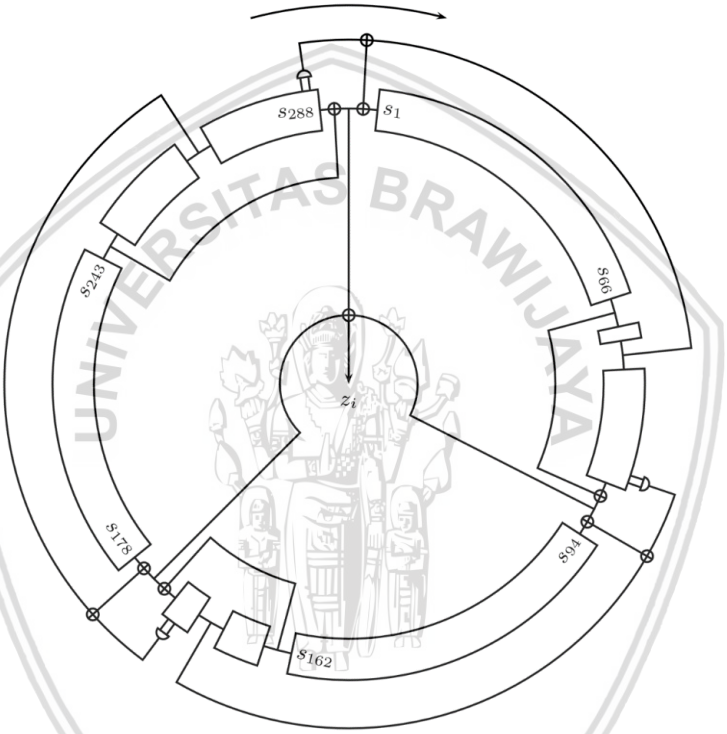
$t_2 \leftarrow S_{162} + S_{175} . S_{176} + S_{177} + S_{264}$

$t_3 \leftarrow S_{243} + S_{286} . S_{287} + S_{288} + S_{69}$

$(S_1, S_2, \ldots, S_{93}) \leftarrow (t_3, S_1, \ldots, S_{92})$

$(S_{94}, S_{95}, \ldots, S_{177}) \leftarrow (t_1, S_{94}, \ldots, S_{176})$

$(S_{178}, S_{279}, \ldots, S_{288}) \leftarrow (t_2, S_{178}, \ldots, S_{287})$



Gambar 2.2 Algoritme Trivium

Gambar 2.2 merupakan penjelasan dari struktur algoritme Trivium. Terdapat 288 bit internal state pada algoritme Trivium yang terdiri dari tiga shift register dengan panjang yang berbeda.

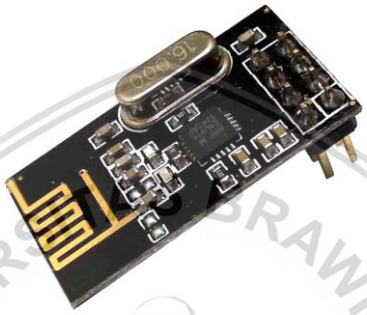
2.2.6 Modul Komunikasi NRF24L01

Modul komunikasi NRF24L01 merupakan modul komunikasi jarak jauh yang menggunakan frekuensi pita gelombang radio 2.4-2.5 GHz ISM (*Industrial Scientific and Medical*). nRF24L01 memiliki kecepatan sampai 2Mbps dengan pilihan opsi *date rate* 250 Kbps, 1 Mbps, dan 2 Mbps. Transceiver terdiri dari synthesizer frekuensi terintegrasi, kekuatan amplifier, osilator kristal, demodulator, modulator dan Enhanced ShockBurst ™ mesin protokol. output daya, saluran frekuensi, dan setup protokol yang mudah diprogram melalui antarmuka SPI. Konsumsi arus yang digunakan sangat rendah, hanya 9.0mA pada

daya output -6dBm dan 12.3mA dalam mode RX. Built-in Power Down dan mode standby membuat penghematan daya dengan mudah realisasi. (Nordic Semiconductor ASA., 2006).

Modul komunikasi NRF24L01 memiliki 8 buah pin diantaranya VCC(3.3V DC), GND, CE, CSN, MOSI, MISO, SCK dan IRQ. Modul ini memiliki beberapa fitur yaitu:

1. Beroperasi pada ISM 2.4 GHZ
2. Kecepatan pengiriman data 250Kbps hingga 2 Mbps
3. Operasi daya ultra rendah.



Gambar 2.3 Modul Komunikasi NRF24L01

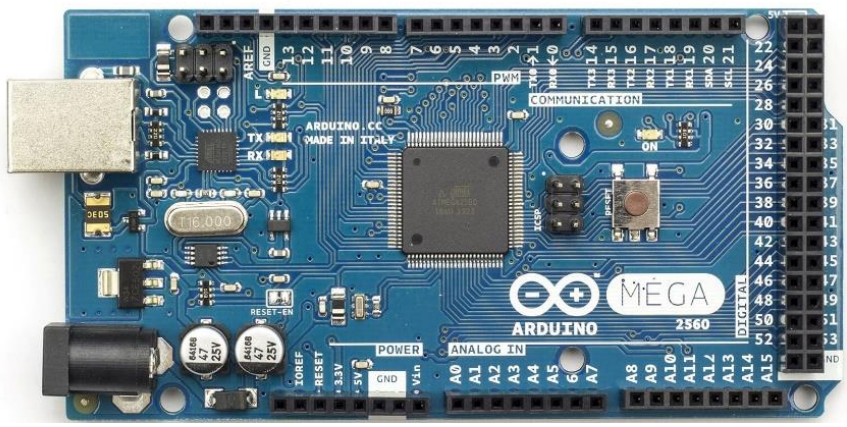
Gambar 2.3 merupakan bentuk dari modul Komunikasi NRF24L01 yang digunakan dalam penelitian ini.

2.2.7 Microcontroller Arduino

Arduino merupakan sebuah *microcontroller single-board computer* yang sering digunakan untuk sering digunakan untuk mengatur ataupun mengontrol suatu sistem yang *embeded*. Arduino dirancang sedemikian rupa sehingga memudahkan para penggunanya dibidang elektronika. *Board* Arduino didesain menggunakan *processor* Atmel AVR dan mendukung masukan dan keluaran pada *board*-nya (Kadir, 2014). Bahasa pemrograman yang digunakan adalah C/C++. Dalam *microcontroller* Arduino dapat ditanamkan berbagai macam *library* maupun metode selama kapasitas memorinya mencukupi.

2.2.7.1 Arduino Mega 2560

Arduino Mega 2560 adalah *board microcontroller* berbasiskan ATmega2560. Arduino Mega 2560 kompatibel dengan sebagian besar *shield* yang dirancang untuk Arduino Duemilanove atau Arduino Diecimila. Arduino Mega 2560 adalah versi terbaru yang menggantikan versi Arduino Mega.



Gambar 2.4 Arduino Mega 2560

Gambar 2.4 merupakan bentuk dari Arduino Mega 2560 yang digunakan dalam penelitian ini.

Arduino Mega 2560 mempunyai beberapa keunggulan yaitu kemampuan untuk menyimpan source yang lebih besar dan jumlah pin digital yang cukup banyak dibandingkan dengan Arduino pada umumnya. Tabel 2.2 akan menjelaskan secara jelas mengenai spesifikasi dari Arduino Mega 2560.

Berikut merupakan spesifikasi Arduino Mega 2560:

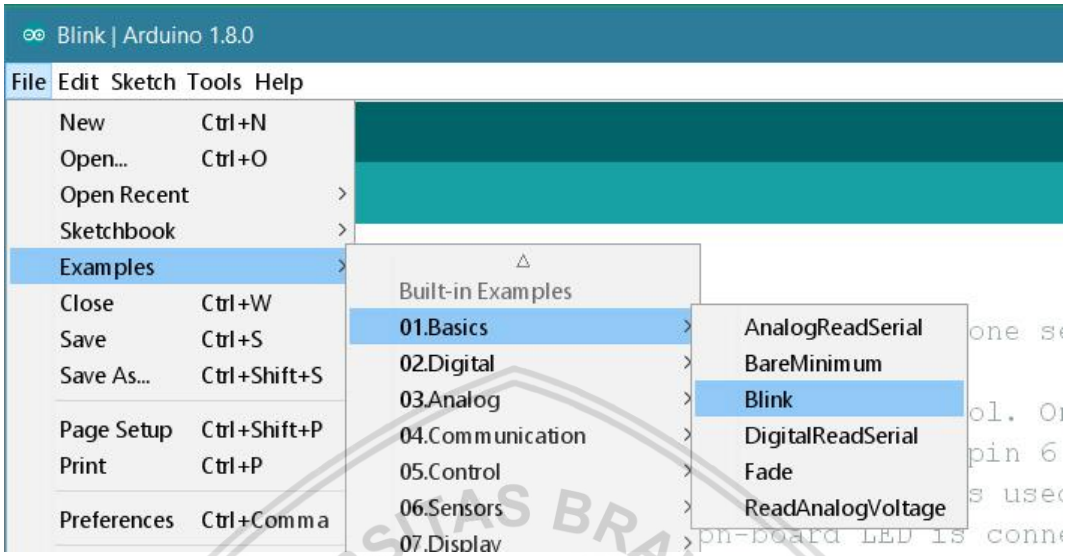
Tabel 2.2 Spesifikasi Arduino Mega 2560

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
USB Host Chip	MAX3421E
Length	101.52 mm
Width	53.3 mm
Weight	36 g

2.2.7.2 Arduino IDE

Arduino IDE merupakan sebuah perangkat lunak yang digunakan sebagai tempat untuk menulis logika-logika dari suatu skema rangkaian yang terhubung dengan *board* Arduino. Arduino IDE dibangun dengan bahasa pemrograman Java

dan bersifat cross-platform (Kadir, 2014) . Barisan kode dalam Arduino IDE ditulis mengikuti aturan dari C/C++ dan baris kode ini disebut dengan istilah *sketch*. Gambar 2-5 merupakan halaman antarmuka Arduino IDE.

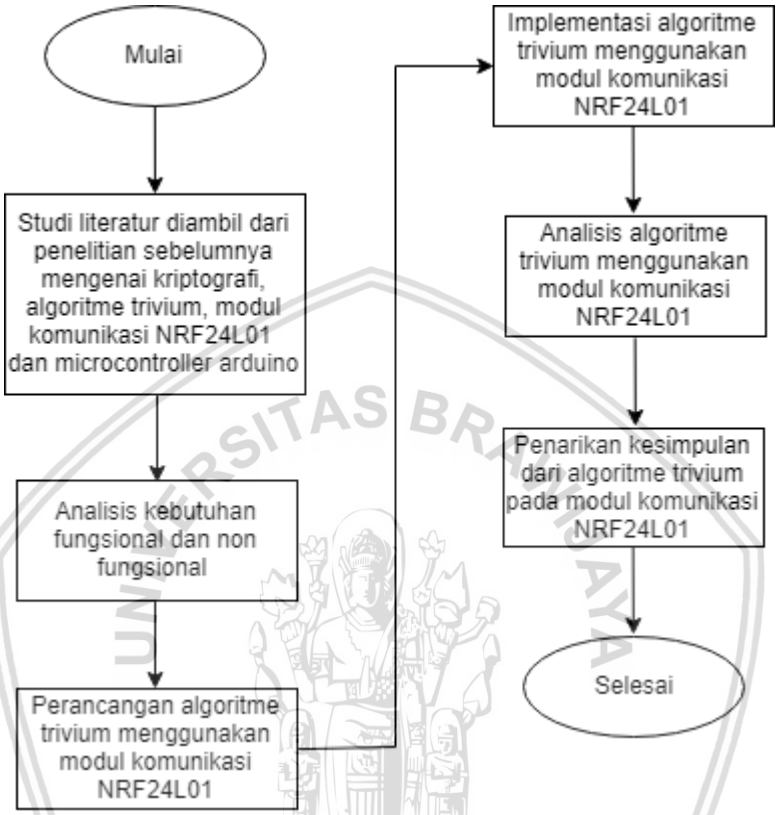


Gambar 2.5 Halaman Antarmuka Arduino IDE

Gambar 2.5 merupakan halaman antarmuka dari Arduino IDE yang akan digunakan untuk memasukkan *sketch* yang akan dijalankan pada *board* Arduino Mega 2560.

BAB 3 METODOLOGI

Pada bab metodologi ini akan membahas mengenai studi literatur, rekayasa kebutuhan sistem baik itu perangkat keras maupun perangkat lunak, perancangan sistem serta implementasi dari sistem.



Gambar 3.1 Diagram Alur Penelitian

Gambar 3.1 menjelaskan mengenai diagram alur penelitian yang akan dilakukan. Hal pertama yang dilakukan yaitu dengan melakukan studi literatur dengan mencari dasar teori yang berkaitan dengan penelitian yang akan dilakukan melalui buku, artikel maupun jurnal. Setelah itu melakukan analisis kebutuhan fungsional dan non-fungsional sistem yang akan dibutuhkan dalam penelitian. Dilanjutkan dengan melakukan perancangan yang akan digunakan sebagai dasar dalam melakukan implementasi algoritme Trivium pada modul komunikasi NRF24L01. Selanjutnya melakukan pengujian dan analisis dari hasil yang diperoleh algoritme Trivium pada modul komunikasi NRF24L01. Dan yang terakhir melakukan penarikan kesimpulan dari penelitian yang dilakukan.

3.1 Studi Literatur

Studi literatur dimulai dengan mengumpulkan teori-teori maupun referensi yang dapat digunakan untuk mendukung penelitian ini. Selanjutnya melakukan analisis kebutuhan baik kebutuhan fungsional maupun kebutuhan non fungsional untuk mendukung penelitian yang akan dilakukan. Lalu melakukan perancangan,

implementasi dan analisis dari penelitian ini. Selanjutnya melakukan penarikan kesimpulan dan saran untuk penelitian selanjutnya.

3.2 Rekayasa Kebutuhan Sistem

Pada sub bab ini menjelaskan mengenai kebutuhan yang akan digunakan dalam melakukan penelitian ini baik dalam segi perangkat keras maupun perangkat lunak agar tujuan yang dari penelitian ini dapat tercapai .

3.2.1 Perangkat Keras

Kebutuhan sistem dari segi perangkat keras yang dibutuhkan dalam melakukan penelitian ini adalah sebagai berikut:

1. Laptop
2. Modul komunikasi NRF24L01
3. Arduino Mega 2560

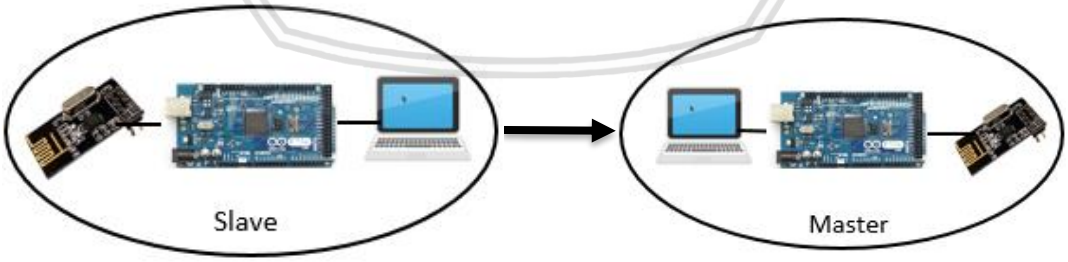
3.2.2 Perangkat Lunak

Kebutuhan sistem dari segi perangkat lunak yang dibutuhkan dalam penelitian ini adalah sebagai berikut:

1. Windows 10 Pro 64-bit
2. Arduino IDE

3.3 Perancangan Sistem

Pada sub bab perancangan sistem akan dijelaskan perancangan sistem dari penelitian ini agar implementasi sistem dapat berjalan dengan baik secara terstruktur dan sistematis. Berikut perancangan sistem dari penelitian yang akan dilakukan:



Gambar 3.2 Perancangan Sistem

Pada gambar 3.2 telihat bahwa sistem memiliki dua bagian *node* yaitu bagian *master* dan bagian *slave.* Dan pada masing-masing bagian *node* sistem terdapat tiga komponen yaitu laptop, Arduino Mega 2560 dan modul komunikasi NRF24L01. Pada modul Arduino Mega 2560 terjadi pemrosesan enkripsi pada

sistem yang sebagai *slave* dan dekripsi pada sistem yang berperan sebagai *master* yang mana nantinya akan diambil berupa variabel waktu untuk menjalankan algoritme Trivium. Selanjutnya NRF24L01 befungsi sebagai sarana komunikasi antar *node master-slave*. Sedangkan pada laptop akan dilakukan konfigurasi mulai dari kebutuhan perangkat lunak hingga konfigurasi algoritme Trivium pada *Key* dan IV yang akan mempengaruhi proses enkripsi dan dekripsi.

3.4 Implementasi Sistem

Implementasi sistem merupakan bagian yang dilakukan berdasarkan kebutuhan sistem dan perancangan sistem yang telah dibuat sebelumnya.

1. Melakukan instalalasi Arduino IDE
2. Melakukan konfigurasi tipe *board* dan komunikasi serial pada Arduino IDE 1.8.5.
3. Menganalisis hasil dari proses enkripsi algoritme Trivium pada *node slave* dan proses dekripsi algoritme Trivium pada *node master*.

3.5 Pengujian dan Analisis Sistem

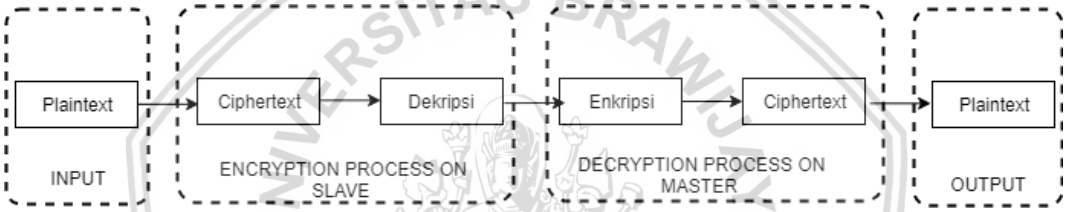
Tahap pengujian sistem dibagi menjadi 4 pengujian, yang masing-masing akan dijelaskan yaitu:

1. Pengujian pada input data bit.
    - Kesesuaian data biner yang dimasukkan dimana terdiri dari 8, 16 dan 32 bit.
    - Data bit yang dimasukkan dapat menempati variabel bit dengan sesuai.
2. Pengujian pada sisi *microcontroller*.
    - Dapat disesuaikannya algoritme *stream cipher* Trivium dengan *microcontroller* Arduino Mega 2560.
    - Dapat menjalankan daya yang dimasukkan pada *microcontroller* Arduino Mega 2560.
3. Pengujian pada sisi NRF24L01.
    - Dapat mengirim data yang akan dikirimkan pada sisi *slave*.
    - Dapat menerima data yang dikirimkan pada sisi master.
4. Pengujian pada data bit *stream cipher*.
    - Kesesuaian hasil dari perubahan plaintext ke ciphertext dengan menggunakan algoritme Trivium.
    - Dapat menampilkan hasil *cipher* pada serial monitor Arduino IDE untuk menentukan kesesuaian dengan *ciphertext*.

BAB 4 REKAYASA KEBUTUHAN

4.1 Gambaran Umum Sistem

Sistem pada penelitian kali ini menggunakan algoritme Trivium yang diimplementasikan untuk mengamankan komunikasi data *master-slave* pada modul komunikasi NRF24L01. Dalam hal ini *node* yang berperan sebagai *slave* melakukan proses enkripsi dan *node* yang berperan sebagai *master* melakukan proses dekripsi. Komunikasi data yang terjadi antara *master* dengan *slave* yaitu *slave* mengirimkan *ciphertext* yang merupakan hasil dari *plaintext* yang telah di enkripsi, selanjutkan *master* menerima *ciphertext* tersebut dan melakukan dekripsi untuk mengubah *ciphertext* tersebut menjadi plaintext kembali. Untuk mendapatkan hasil *ciphertext* dilakukan dengan mengolah *plaintext* bersama dengan *keystream.* Dan untuk mengubah kembali *ciphertext* kepada bentuk pesan semula, yaitu dengan mengolah *ciphertext* dengan *keystream*. Berikut merupakan gambaran blok system.



Gambar 4.1 Diagram Blok Sistem

Penelitian ini bertujuan untuk membuat sistem keamanan komunikasi data ada sisi *hardware*. Dan untuk mengetahui performa *hardware* saat menjalankan sistem, maka dilakukan pengujian parameter performa pemrosesan *keystream*, performa saat melakukan proses enkripsi maupun proses dekripsi dan performa pengiriman hasil enkripsi dari *slave* ke *master*.

4.2 Kebutuhan Sistem

Pada sub bab kebutuhan sistem ini, akan menjelaskan kebutuhan fungsional, dan kebutuhan non-fungsional. Selainn itu berisi juga penjelasan mengenai kebutuhan perangkat keras dan perangkat lunak yang akan dibutuhkan dalam penelitian ini.

4.2.1 Kebutuhan Fungsional

Terdapat beberapa kebutuhan fungsional yang dibutuhkan dalam penelitian ini yaitu:

1. Mikrokontroller yang harus mampu memproses *key* dan *initial value* agar menjadi keystream.
2. Modul komunikasi yang mampu mengirimkan data *ciphertext* dari *master* ke *slave*.

3. Masukan enkripsi pada master dan keluaran dekripsi pada slave harus sesuai.

4.2.2 Kebutuhan Non Fungsional

Kebutuhan non fungsional yang dibutuhkan dalam penelitian ini yaitu:

1. Pemberian tegangan sebesar 3,3V pada mikrokontroller Arduino Mega5260, agar proses enkripsi dan dekripsi yang terjadi tetap stabil.

4.2.3 Kebutuhan Perangkat Keras

Berikut beberapa kebutuhan perangkat keras yang dibutuhkan dalam sistem ini:

1. Komputer/Laptop

   Spesifikasi laptop yang digunakan dalam sistem ini yaitu:

   - Model: Asus-WX028D
   - Manufaktur: ASUSTek Computer Inc.
   - Processor: Intel(R) Core(TM) i5-4200U CPU @1.60GHz 2.30ghz
   - RAM: 4GB

2. Arduino Mega 2560

   Arduino Mega 2560 merupakan mikrokontroller yang digunakan untuk melakukan proses enkripsi pada *master* dan proses dekripsi pada *slave*.

3. Modul komunikasi NRF24L01

   NRF24L01 merupakan modul komunikasi data yang digunakan untuk mengirimkan dan menerima paket data yang telah di enkripsi.

4.2.4 Kebutuhan Perangkat Lunak

Berikut merupakan beberapa kebutuhan perangkat lunak yang dibutuhkan dalam sistem ini:

1. Windows 10 Pro 64-bit

   Windows 10 Pro 64-bit merupakan sistem operasi yang digunakan pada laptop yang digunakan untuk sistem ini.

2. Arduino IDE

   Arduino IDE merupakan perangkat lunak yang digunakan untuk melakukan *programming* pada Arduino Mega 2560 serta menampilkan keluaran dari Arduino Mega 2560 pada serial monitor Arduino IDE.

BAB 5 PERANCANGAN DAN IMPLEMENTASI

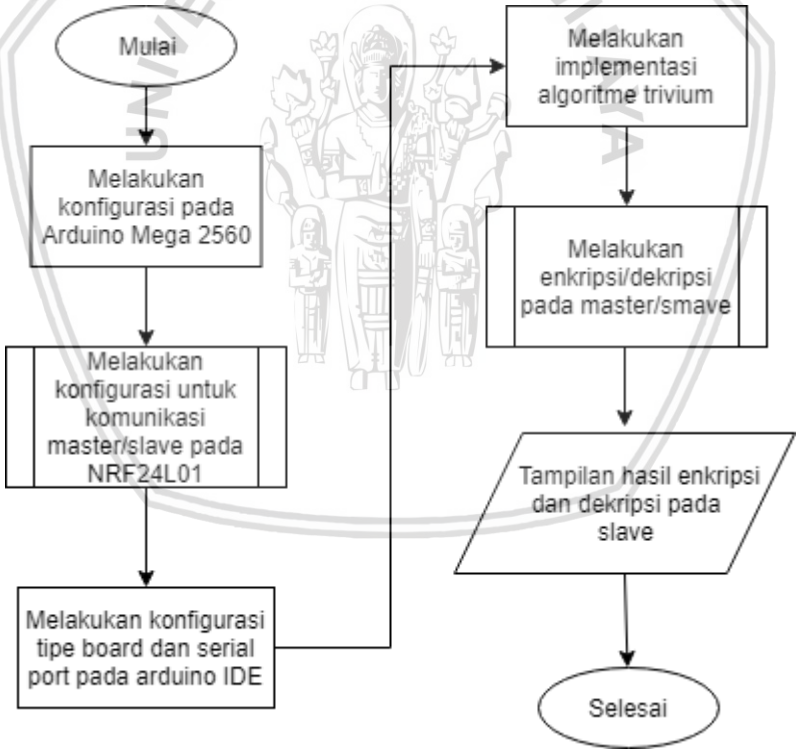
Pada bab lima ini akan menjelaskan tentang perancangan sistem yang akan dilakukan sebagai dasar dari implementasi sistem dan dilanjutkan dengan penjelasan mengenai implementasi sistem yang akan dilakukan pada penelitian ini.

5.1 Perancangan

Perancangan sistem merupakan bagian penting berupa perencanaan secara teknis dari sistem yang dibuat agar kebutuhan sistem dapat diimplementasikan dan dijalankan dengan benar.

5.1.1 Diagram Alir Perancangan Sistem

Pada sub bab ini akan dijelaskan mengenai tahapan dari perancangan sistem yang sesuai pada gambaran umum sistem pada bab sebelumnya. Gambar 5.1 merupakan diagram alir dari perancangan sistem:



Gambar 5.1 Diagram Alir Perancangan Sistem

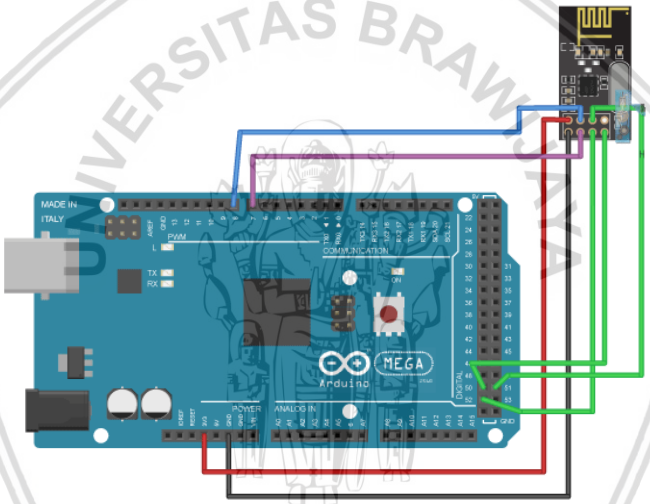
Pada Gambar 5.1 menjelaskan perancangan sistem yang dimulai dengan melakukan konfigurasi pada Arduino Mega 2560. Selanjutnya melakukan konfigurasi untuk komunikasi NRF24L01 agar komunikasi yang terjadi antara *master* dengan *slave* dapat berjalan dengan benar.Lalu dilanjutkan dengan

konfigurasi tipe *board* sesuai dengan Arduino Mega 2560 dan menyesuaikan serial port yang digunakan oleh masing-masing *master* dan *slave.* Selanjutnya melakukan implementasi algoritme Trivium dengan melakukan proses enkripsi-dekripsi pada *master-slave*. Dan yang terakhir *slave* menampilkan hasil enkripsi yang diterima dari *master* serta hasil dekripsi yang telah dilakukan.

5.1.2 Perancangan Perangkat Keras

Dalam sistem yang akan dibuat, komunikasi antara *master* dengan *slave* merupakan gabungan dari mikrokontroller Arduino Mega 2560 dengan modul komunikasi NRF24L01. Berikut merupakan jumlah perangkat keras yang dibutuhkan untuk membuat sistem adalah sebagai berikut:

- Duah buah mikrokontroller Arduino Mega 2560
- Dua buah modul komunikasi NRF24L01
- Tujuh buah kabel jumper



Gambar 5.2 Rangkaian Perancangan Perangkat keras

Gambar 5.2 menunjukkan rangkaian perancangan perangkat keras yang akan dibuat pada sistem. Perangkat mikrokontroller Arduino Mega 2560 dan modul komunikasi NRF24L01 dihubungkan menggunakan kabel jumper. Untuk *pin* yang menghubungkan Arduino Mega 2560 dengan modul komunikasi NRF24L01 akan dijelaskan pada table 5.1.

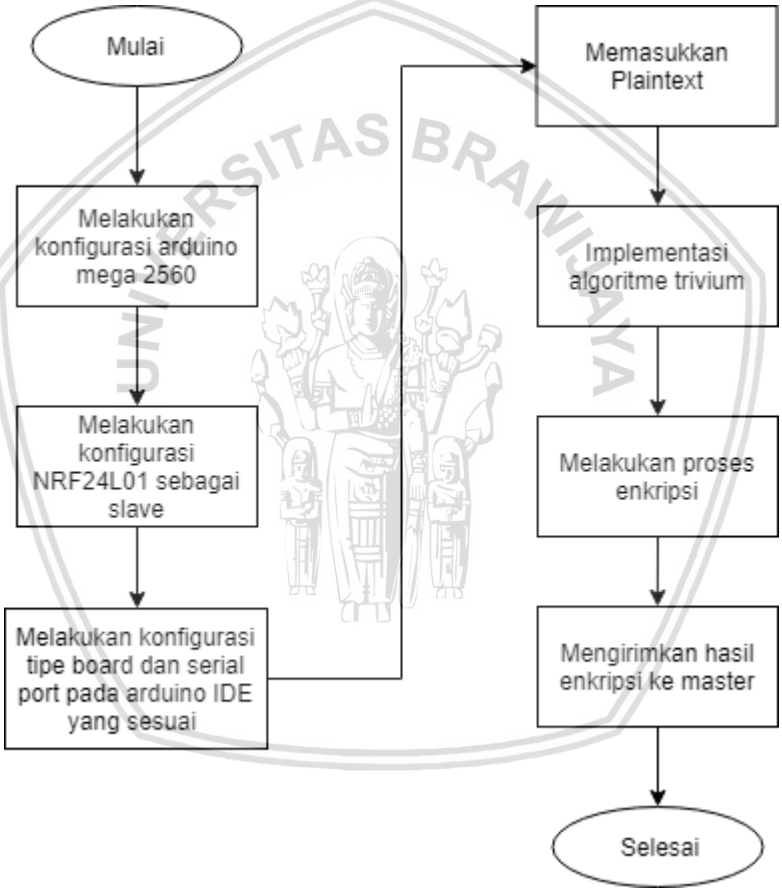
Tabel 5.1 Koneksi Pin Arduino Mega 2560 Dengan Modul NRF24L01

Arduino Mega 2560	NRF24L01
GND	GND
3.3V	VCC
9	CE
53	CSN
52	SCK
51	MOSI
50	MISO

Rangkaian pada Tabel 5.1, mikrokontroler Arduino Mega 2560 berfungsi sebagai 'otak' sistem dari *master* maupun *slave*. Mikrokontroler Arduino Mega 2560 berfungsi untuk melakukan proses enkripsi pada *master* dan proses dekripsi pada *slave* serta mengatur pengiriman dan penerimaan data. Sedangkan modul komunikasi NRF24L01 hanya berfungsi sebagai media komunikasi antara *master* dengan *slave*.

5.1.3 Enkripsi Pada Slave

Pada sub bab ini akan dijelaskan lebih mendalam mengenai proses enkripsi yang dilakukan oleh *slave.* Berikut adalah *flowchart* perancangan dari proses enkripsi:

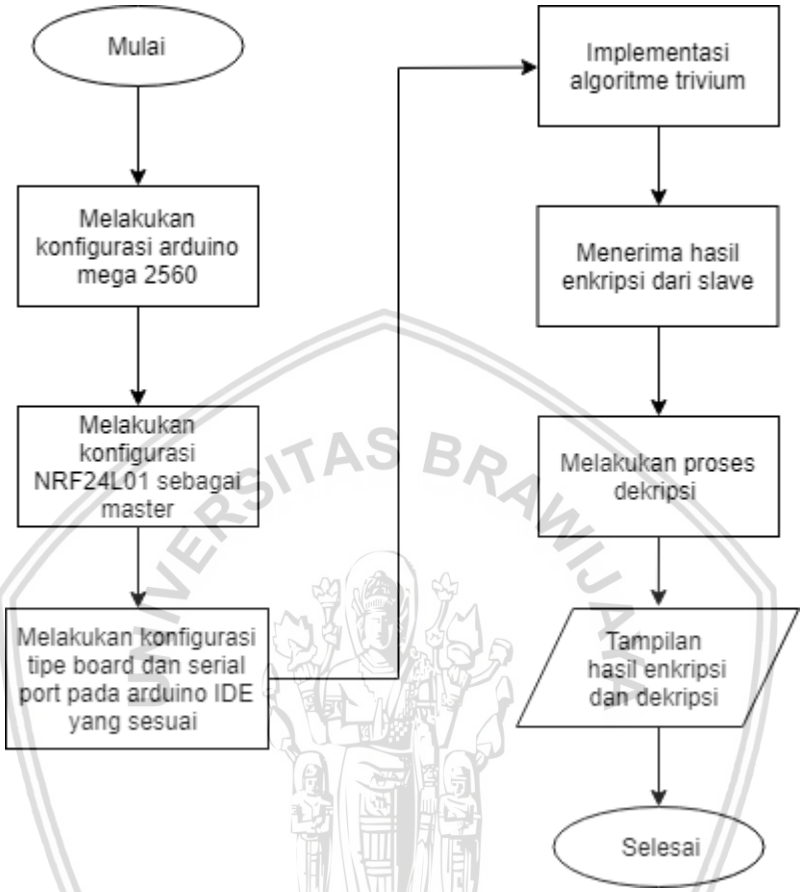


Gambar 5.3 Proses Enkripsi Pada *Slave*

Gambar 5.2 menjelaskan tahapan-tahapan untuk melakukan proses enkripsi dimana proses enkripsi dilakukan oleh *slave.* Dimulai dengan konfigurasi pada Arduino Mega 2560 dan konfigurasi NRF24L01 sebagai *slave* serta konfigurasi tipe *board* dan serial port yang sesuai pada Arduino IDE 1.8.5. Selanjutnya melakukan implementasi dari algoritme Trivium dan melakukan proses enkripsi dari masukan plaintext serta mengirimkan hasil dari enkripsi kepada *master.*

5.1.4 Dekripsi Pada Master

Pada sub bab ini juga akan menjelaskan lebih mendalam mengenai proses dekripsi yang dilakukan oleh *master*. Dan berikut ini merupakan *flowchart* rancangan dari proses dekripsi pada *master*:



Gambar 5.4 Proses Dekripsi Pada *Master*

Gambar 5.3 menjelaskan mengenai tahapan-tahapan yang dilakukan oleh *master* untuk melakukan proses dekripsi. Hal yang pertama dilakukan hampir sama seperti *slave* yaitu melakukan konfigurasi pada Arduino Mega 2560 dan konfigurasi NRF24L01 sebagai *master* serta konfigurasi tipe *board* dan serial port yang sesuai pada Arduino IDE 1.8.5. Selanjutnya mengimplementasikan algoritme Trivium dan menerima hasil enkripsi dari *slave*. Setelah itu, *master* melakukan proses dekripsi untuk mengembalikan pesan ke bentuk awal. Setelah selesai melakukan dekripsi, slave menampilkan hasil enkripsi yang telah diterima dari *slave* dan hasil dekripsi yang telah dilakukan.

5.2 Implementasi

Pada bagian ini akan membahas mengenai implementasi dari perancangan sistem yang telah dijelaskan sebelumnya. Cakupan implementasi meliputi bagaimana penerapan algoritme Trivium dalam mengamankan data *master-slave* pada perangkat NRF24L01.

5.2.1 Proses Enkripsi Pada Slave

Proses enkripsi pada master dilakukan sesuai dengan tahapan yang telah dijelaskan pada bagian perancangan sistem.

5.2.1.1 Key and IV Setup Slave

Algoritme Trivium ini diinisialisasikan dengan memasukkan 80 bit kunci dan 80 bit IV(*Initial Value*) kedalam *internal state* yang berukuran 288 bit . Pada proses key and IV setup terdiri dari dua tahapan dimana masing-masing tahapan akan dijelaskan secara berbeda.

Tabel 5.2 *Source code* Pengisian *Internal State* pada *Slave*

```
1   Void key_iv_setup(){
2   for(i=0; i<93; i++){
3      if(i < 80)
4        states[i] = key[i];
5      else
6        states[i] = 0;
7    }
8
9    for(i=93; i<177; i++){
10     if((i-93) < 80)
11       states[i] = iv[i-93];
12     else
13       states[i] = 0;
14   }
15   for(i=177; i<288; i++){
16     if(i<285)
17       states[i] = 0;
18     else
19       states[i] = 1;
20   }
21   }
```

Tabel 5.2 menjelaskan mengenai proses pengisian *internal state* 288 bit dengan menggunakan *key* dan IV yang ada. *Internal state* di bagi menjadi tiga register dengan masing-masing panjang *state* 93,84 dan 111. Untuk *internal state register* pertama diisi dengan *key* 80 bit pertama dan sisa bit *state* diisi dengan 0. Dan untuk *internal state register* yang kedua diisi dengan IV 80 bit pertama dan sisa *state* juga diisi dengan 0. Sedangkan untuk *internal state register* yang ketiga mengisi nilai bit *state* dengan 0 kecuali untuk *internal state* ke 286, 287 dan 288.

Tabel 5.3 Source code Initial State Pada Slave

```
1  void init_state(){
2    for(j=0; j<(4*288); j++){
3  t1  =  states[65]  ^  (states[90]  &  states[91])  ^
4  states[92] ^ states[170];
5  t2  =  states[161]  ^  (states[174]  &  states[175])  ^
6  states[176] ^ states[263];
```

```
7   t3 = states[242] ^ (states[285] & states[286]) ^
8   states[287] ^ states[68];
9       for(i=0; i<93; i++){
10      if(i == 0){
11        temp = states[i];
12        states[i] = t3;
13      }
14        else{
15        tempx = states[i];
16        states[i] = temp;
17        temp = tempx;
18      }
19      }
20      for(i=93; i<177; i++){
21      if(i == 93){
22        temp = states[i];
23        states[i] = t1;
24      }
25        else{
26        tempx = states[i];
27        states[i] = temp;
28        temp = tempx;
29      }
30      }
31      for(i=177; i<288; i++){
32      if(i == 177){
33        temp = states[i];
34        states[i] = t2;
35      }
36        else{
37        tempx = states[i];
38        states[i] = temp;
39        temp = tempx;
40      }
41      }
42                                }
```

Tabel 5.3 menjelaskan mengenai proses pengisian *state* awal dengan melakukan rotasi *state* sebanyak 1152 kali dan melakukan pergeseran bit.Selanjutnya hasil dari rotasi yang ke 1152 yang akan digunakan sebagai *internal state* untuk proses *keystream generator*.

5.2.1.2 Keystream Generator Slave

Proses *keystream generator* mengandung proses perulangan yang mengekstrak nilai dari 15 bit spesifik *state* dan menggunakannya untuk mengperbarui tiga bit dari *state* dan menghitung 1 bit dari *keystream zi.* Bit-bit state kemudian dirotasikan dan proses berulang hingga $N \leq 2^{64}$ *keystream* yang dibutuhkan. Dan untuk implementasi dari *keystream generator* akan dijelaskan pada table 5.4.

Tabel 5.4 *source code Keystream Generator* pada *Slave*

```
1   Void key_gen(){
2   for(j=0; j<NUM_BITS_PT; j++){
3       t1 = states[65] ^ states[92];
4       t2 = states[161] ^ states[176];
5       t3 = states[242] ^ states[287];
6
7       z[j] = t1 ^ t2 ^ t3;
8       ciphertext[j] = plain_text[j] ^ z[j];
9
10   t1 ^= (states[90]&states[91])^states[170];
11   t2 ^= (states[174]&states[175])^states[263];
12   t3 ^= (states[285]&states[286])^states[68];
13      for(i=0; i<93; i++){
14      if(i == 0){
15        temp = states[i];
16        states[i] = t3;
17      }
18        else{
19        tempx = states[i];
20        states[i] = temp;
21        temp = tempx;
22      }
23      }
24      for(i=93; i<177; i++){
25      if(i == 93){
26        temp = states[i];
27        states[i] = t1;
28      }
29        else{
30        tempx = states[i];
31        states[i] = temp;
32        temp = tempx;
33      }
34      }
34      for(i=177; i<288; i++){
36      if(i == 177){
37        temp = states[i];
38        states[i] = t2;
39      }
40        else{
41        tempx = states[i];
42        states[i] = temp;
43        temp = tempx;
44      }
45      }
46    }
47    }
48
```

Tabel 5.4 menjelaskan mengenai proses menghasilkan *keystream* yang dibutuhkan. Panjang dari bit *keystream* yang dihasilkan sesuai dengan panjang

bit *plaintext* yang akan di enkripsi. Selain itu pada proses *keystream generator* ini terjadi proses enkripsi dimana per bit *plaintext* di-XOR-kan dengan *zi* sebelum dilakukan rotasi kembali.

5.2.1.3 Mengirimkan Hasil Enkripsi dari Slave

Setelah melakukan konfigurasi Arduino Mega 2560 dan modul komunikasi NRF24L01 dengan benar serta melakukan proses enkripsi, berikut merupakan penjelasan mengenai pengiriman hasil enkripsi menggunakan modul komunikasi NRF24L01.

Tabel 5.5 *Source code* Pengiriman Hasil Enkripsi oleh *Slave*

```
1     for(j=0; j<NUM_BITS_PT; j++){
2   radio.write(&ciphertext,sizeof(ciphertext));
3     }
```

Tabel 5.5 menjelaskan mengenai proses pengiriman yang dilakukan oleh master yaitu dengan menggunakan kode program `radio.write()`. Pengiriman yang dilakukan sesuai dengan banyaknya jumlah ciphertext yang telah dihasilkan.

5.2.2 Proses Dekripsi Pada master

Proses dekripsi yang dilakukan oleh slave dilakukan sesuai dengan tahapan yang telah dijelaskan pada bagian perancangan sistem.

5.2.2.1 Key and IV Setup Master

Key and IV setup yang terjadi pada *master* tidak berbeda dengan yang dilakukan oleh *slave*. *Master* juga melakukan inisialisasi dengan memasukkan 80 bit kunci dan 80 bit IV(*Initial Value*) kedalam *internal state* yang berukuran 288 bit . Pada proses key and IV setup terdiri dari dua tahapan dimana masing-masing tahapan akan dijelaskan secara berbeda.

Tabel 5.6 *Source code* Pengisian *Internal State* pada *Master*

```
1   Void key_iv_setup(){
2    for(i=0; i<93; i++){
3       if(i < 80)
4         states[i] = key[i];
5       else
6         states[i] = 0;
7     }
8
9     for(i=93; i<177; i++){
10      if((i-93) < 80)
11        states[i] = iv[i-93];
12      else
13        states[i] = 0;
14    }
15    for(i=177; i<288; i++){
16      if(i<285)
```

```
17        states[i] = 0;
18      else
19        states[i] = 1;
20    }
21                                }
```

Tabel 5.6 menjelaskan mengenai proses pengisian *internal state* 288 bit dengan menggunakan *key* dan IV yang ada. *Internal state* di bagi menjadi tiga register dengan masing-masing panjang *state* 93,84 dan 111. Untuk *internal state register* pertama diisi dengan *key* 80 bit pertama dan sisa bit *state* diisi dengan 0. Dan untuk *internal state register* yang kedua diisi dengan IV 80 bit pertama dan sisa *state* juga diisi dengan 0. Sedangkan untuk *internal state register* yang ketiga mengisi nilai bit *state* dengan 0 kecuali untuk *internal state* ke 286, 287 dan 288.

Tabel 5.7 *Source code Initial State* pada *Master*

```
1   void init_state(){
2     for(j=0; j<(4*288); j++){
3   t1  =  states[65]  ^  (states[90]  &  states[91])  ^
4   states[92] ^ states[170];
5   t2  =  states[161]  ^  (states[174]  &  states[175])  ^
6   states[176] ^ states[263];
7   t3  =  states[242]  ^  (states[285]  &  states[286])  ^
8   states[287] ^ states[68];
9       for(i=0; i<93; i++){
10      if(i == 0){
11        temp = states[i];
12        states[i] = t3;
13      }
14        else{
15        tempx = states[i];
16        states[i] = temp;
17        temp = tempx;
18      }
19      }
20      for(i=93; i<177; i++){
21      if(i == 93){
22        temp = states[i];
23        states[i] = t1;
24      }
25        else{
26        tempx = states[i];
27        states[i] = temp;
28        temp = tempx;
29      }
30      }
31      for(i=177; i<288; i++){
32      if(i == 177){
33        temp = states[i];
34        states[i] = t2;
35      }
```

```
36          else{
37          tempx = states[i];
38          states[i] = temp;
39          temp = tempx;
40        }
41        }
42  }
```

Tabel 5.7 menjelaskan mengenai proses pengisian *state* awal dengan melakukan rotasi *state* sebanyak 1152 kali dan melakukan pergeseran bit. Selanjutnya hasil dari rotasi yang ke 1152 yang akan digunakan sebagai *internal state* untuk proses *keystream generator*.

5.2.2.2 Keystream Generator Master

Proses *keystream generator* pada slave juga mengandung proses perulangan yang mengekstrak nilai dari 15 bit spesifik *state* dan menggunakannya untuk mengperbarui tiga bit dari *state* dan menghitung 1 bit dari *keystream zi.* Bit-bit state kemudian dirotasikan dan proses berulang hingga $N \leq 2^{64}$ *keystream* yang dibutuhkan. Dan untuk implementasi dari *keystream generator* akan dijelaskan pada table 5-8.

Tabel 5.8 *Source code Keystream Generator* pada *Master*

```
1   Void key_gen(){
2   for(j=0; j<NUM_BITS_PT; j++){
3       t1 = states[65] ^ states[92];
4       t2 = states[161] ^ states[176];
5       t3 = states[242] ^ states[287];
6
7       z[j] = t1 ^ t2 ^ t3;
8
9    t1 ^= (states[90]&states[91])^states[170];
10   t2 ^= (states[174]&states[175])^states[263];
11   t3 ^= (states[285]&states[286])^states[68];
12      for(i=0; i<93; i++){
13      if(i == 0){
14        temp = states[i];
15        states[i] = t3;
16      }
17        else{
18        tempx = states[i];
19        states[i] = temp;
20        temp = tempx;
21      }
22      }
23      for(i=93; i<177; i++){
24      if(i == 93){
25        temp = states[i];
26        states[i] = t1;
27      }
```

```
28          else{
29          tempx = states[i];
30          states[i] = temp;
31          temp = tempx;
32        }
33        }
34        for(i=177; i<288; i++){
34        if(i == 177){
36          temp = states[i];
37          states[i] = t2;
38        }
39          else{
40          tempx = states[i];
41          states[i] = temp;
42          temp = tempx;
43        }
44        }
45      }
46      }
```

Tabel 5.4 menjelaskan mengenai proses menghasilkan *keystream* yang dibutuhkan oleh *master*. Panjang dari bit *keystream* yang dihasilkan sesuai dengan panjang bit *plaintext* yang akan di dekripsi. Perbedaaan antara proses menghasilkan *keystream* pada *slave* dengan menghasilkan *keystream* pada *master* yaitu tidak adanya operasi XOR yang dilakukan sehingga *master* hanya menghasilkan nilai *zi*.

5.2.2.3 Menerima Hasil Enkripsi Pada Master

Setelah melakukan konfigurasi Arduino Mega 2560 dan modul komunikasi NRF24L01 untuk sebagai *master* dengan benar serta melakukan proses menghasilkan *keystream* dengan menggunakan algoritme Trivium, berikut merupakan penjelasan mengenai proses penerimaan hasil enkripsi menggunakan modul komunikasi NRF24L01 oleh *master*.

Tabel 5.9 *Source code* Penerimaan Hasil Enkripsi oleh *Master*

```
1   if (radio.available()) {
2    radio.read(&ciphertext,sizeof(ciphertext));
3   }
```

Tabel 5.9 menjelaskan mengenai proses penerimaan hasil enkripsi yang dilakukan oleh *master* yaitu dengan menggunakan kode program `if(radio.available())` untuk memastikan *master* dengan *slave* telah terhubung. Setelah *master* dan *slave* telah terhubung maka pesan yang dikirimkan oleh *slave* akan diterima oleh *master* dengan menggunakan kode program `radio.read()`.

5.2.2.4 Menampilkan Hasil Enkripsi dan Dekripsi pada *Master*

Untuk menampilkan hasil enkripsi dan hasil dekripsi yang benar, tahapan mulai dari konfigurasi hingga proses menghasilkan dekripsi harus telah selesai dilakukan. Pada Tabel 5.10 akan menjelaskan proses menampilkan hasil enkripsi dan dekripsi pada *master*.

Tabel 5.10 *Source code* Menampilkan Hasil Enkripsi dan Dekripsi pada Master

```
1   Serial.print("\nHasil Enkripsi:    ");
2     for(j=0; j<NUM_BITS_PT/4; j++){
3   ct[j]=ciphertext[4*j]*(8)+ciphertext[4*j+1]*(4)+
4   ciphertext[4*j+2]*(2) + ciphertext[4*j+3]*(1);
5     Serial.print(ct[j], BIN);
6     delay(20);
7     }
8
9     for(i=0; i<NUM_BITS_PT; i++)
10        rec_plain_text[i] = ciphertext[i] ^ z[i];
11
12  Serial.print("Hasil Dekripsi:    ");
13      for(j=0; j<NUM_BITS_PT/4; j++){
14  r_pt[j]=rec_plain_text[4*j]*(8)+rec_plain_text[4*j+1]*(
15  4)+rec_plain_text[4*j+2]*(2)+rec_plain_text[4*j+3]*(1);
16          Serial.print(r_pt[j], BIN);
17          delay(20);
18      }
```

Tabel 5.10 menjelaskan mengenai tahapan dalam menampilkan hasil enkripsi dan hasil dekripsi yang dilakukan oleh *master*. Setelah master menerima hasil enkripsi yang dikirimkan oleh *slave*, maka *master* menampilkan hasil enkripsi tersebut. Selain itu *master* juga akan menggunakan hasil enkripsi yang dikirimkan oleh *slave* untuk melakukan proses dekripsi serta menampilkan hasil dekripsi.

BAB 6 PENGUJIAN DAN ANALISIS

6.1 Parameter Pengujian

Terdapat beberapa parameter yang akan gunakan untuk pengujian kinerja Algoritme Trivium dalam mengamankan data menggunakan modul komunikasi NRF24L01 yaitu:

1. Uji validitas menggunakan *test vector*.
2. Uji waktu yang dibutuhkan untuk memproses *keystream*.
3. Uji keamanan dengan melakukan *sniffing*.
4. Uji waktu yang dibutuhkan untuk memperoleh hasil enkripsi dan dekripsi.
5. Uji waktu yang dibutuhkan untuk mengirimkan hasil enkripsi dari *master* ke *slave* menggunakan modul komunikasi NRF24L01.

6.2 Uji Validitas Menggunakan Test Vector

Pengujian validitas test vector digunakan untuk memastikan bahwa *keystream* yang dihasilkan telah sesuai dengan *test vector*.

6.2.1 Prosedur Pengujian

Prosedur pengujian *test vector* pada algoritme Trivium menggunakan *KEY* dan IV yang ada pada *test vector*, selanjutnya *keystream* yang dihasilkan dari sistem dicocokkan dengan *test vector*.

6.2.2 Hasil Pengujian

Pada sub bab telah dijelaskan tujuan dari melakukan validasi keluaran keystream dari sistem. Tabel 6.1 merupakan hasil dari pengujian validitas *test vector*.

Tabel 6.1 Pengujian Validitas Test Vector

No	KEY	IV	Keystream Sistem	Test vector	Valid/ tidak
1	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 10000000	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	000111001101011 101100001111111 111100111010110 000010111100011 100111110101101 100011000111101 011100001000100 000010000101010 101100001101110 111011110101001	000111001101011 101100001111111 111100111010110 000010111100011 100111110101101 100011000111101 011100001000100 000010000101010 101100001101110 111011110101001	Valid

No	KEY	IV	Keystream Sistem	Test vector	Valid/ tidak
			100010100000100 110000100111010 111010000000101 01010	100010100000100 110000100111010 111010000000101 01010	

Pada tabel 6.1 menjelaskan bahwa nilai *keystream* yang dihasilkan dari sistem sesuai dengan *test vector* sehingga pengujian validitas test vector dapat dinyatakan valid.

6.3 Pengujian Pemrosesan Keystream

Setelah pengujian validitas *test vector* diperoleh, selanjutnya akan dilakukan pengujian waktu pemrosesan untuk menghasilkan *keystream*. Proses pengujian ini bertujuan memperoleh kesimpulan dari hasil pengujian.

6.3.1 Prosedur Pengujian

Dalam melakukan pengujian waktu yang dibutuhkan untuk memperoleh hasil *keystream*, nilai dari *key* dan IV diganti secara manual.

6.3.2 Hasil Pengujian

Tabel 6.2 merupakan hasil dari pengujian waktu pemrosesan untuk menghasilkan *keystream*.

Tabel 6.2 Pengujian Waktu Pemrosesan Keystream

No	KEY	IV	Waktu(mili second)
1	00000000000000000000000000000000000000000000000000000000000000000010000000	00000000000000000000000000000000000000000000000000000000000000000000000000	526
2	00000000000000000000000000000000000000000000000000000000000000001000000000	00000000000000000000000000000000000000000000000000000000000000000000000000	482
3	00000000000000000000000000000000000000000000000000000000000010000000000000	00000000000000000000000000000000000000000000000000000000000000000000000000	427
4	00000000000000000000000000000000000000000000000000001000000000000000000000	00000000000000000000000000000000000000000000000000000000000000000000000000	667

No	KEY	IV	Waktu(mili second)
5	00000000000000000000000000000000000000000000000000000000000000000000000000000000	00000000000000000000000000000000000000000000000000000000000000000000000000000000	421

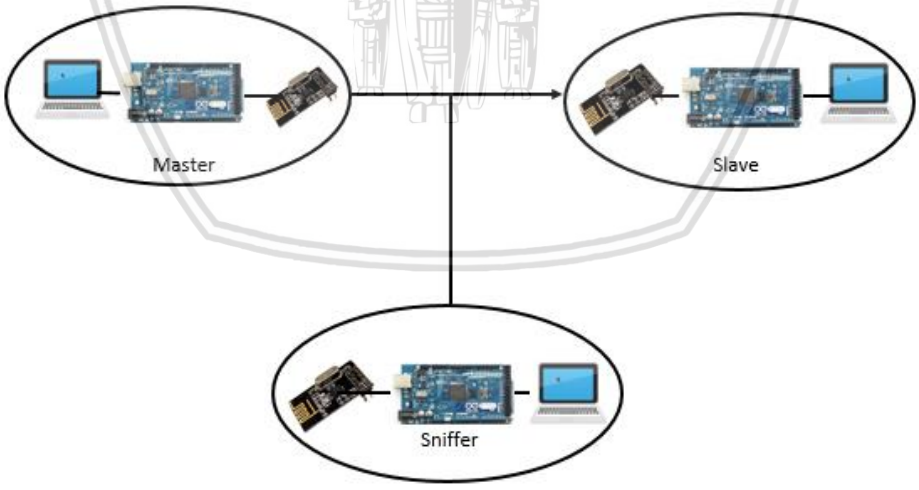
Tabel 6.2 menunjukkan hasil yang diperoleh dari hasil pengujian waktu pemrosesan dari *keystream* dengan mengunakan *key* dan IV yang berbeda. Hasil dari pengujian menunjukan bahwa tidak ada perbedaan yang terlalu signifikan dalam menghasilkan *keystream* dengan menggunakan key dan IV yang berbeda.

6.4 Uji Keamanan Dengan Melakukan Sniffing

Pengujian keamanan dengan melakukan *sniffing* bertujuan untuk memastikan fungsionalitas dari sistem telah berjalan dengan baik. Pengujian ini bertujuan untuk memperoleh kesimpulan dari hasil pengujian.

6.4.1 Prosedur Pengujian

Dalam pengujian keamanan dengan melakukan *sniffing* ini, hanya akan menggunakan paket data berukuran 32 bit. Pengujian dilakukan dengan menambahkan satu node sebagai *sniffer* yang betujuan untuk melakukan *sniffing*. Gambar 6.4 merupakan penjelasan dari topologi pengujian keamanan dengan *sniffing* yang akan dilakukan.



Gambar 6.1 Topologi pengujian keamanan dengan melakukan *sniffing*

Pada gambar 6.4 menjelaskan mengenai topologi pengujian keamanan dengan melakukan *sniffing*. Pada saat komunikasi data antara *master* dengan *slave* berlangsung, *sniffer* akan menyadap komunikasi yang terjadi antara *master* dengan *slave*.

6.4.2 Hasil Pengujian

Tabel 6.3 Nilai Dari Masukan *Plaintext* dan Hasil *Ciphertext*

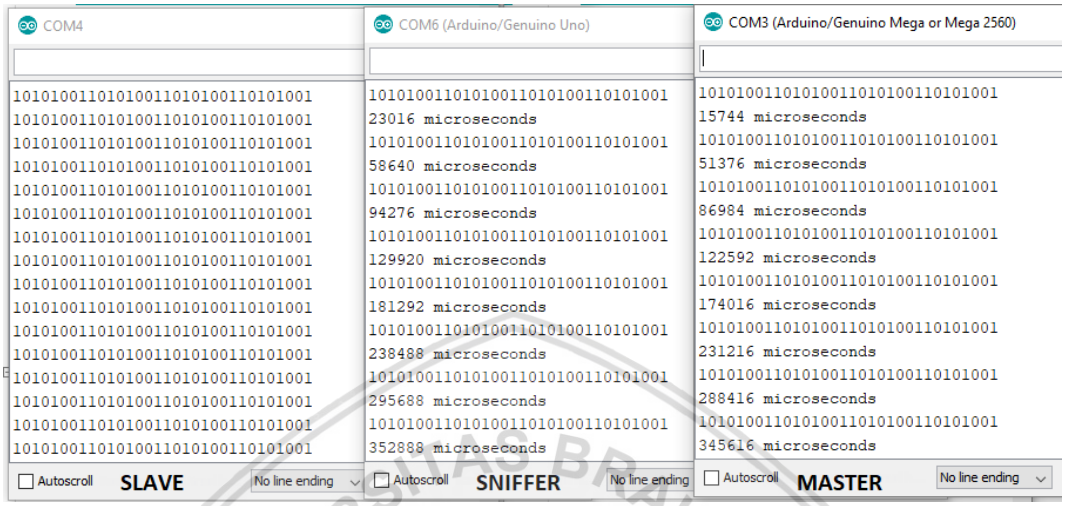
No	Plaintext		Ciphertext
	Hexadecimal	Biner	
1	a9a9a9a9	10101001101010011010100110101001	01110110101011100101010011001101
2	2f2f2f2f	00101111001011110010111100101111	11110000001010001101001001001011
3	00000000	00000000000000000000000000000000	11011111000001111111110101100100
4	11111111	00010001000100010001000100010001	11001110000101101110110001110101
5	22222222	00100010001000100010001000100010	11111101001001011101111101000110
6	33333333	00110011001100110011001100110011	11101100001101001100111001010111
7	44444444	01000100010001000100010001000100	10011011010000111011100100100000
8	55555555	01010101010101010101010101010101	10001010010100101010100000110001
9	66666666	01100110011001100110011001100110	10111001011000011001101100000010
10	77777777	01110111011101110111011101110111	10101000011100001000101000010011
11	88888888	10001000100010001000100010001000	01010111100011110111010111101100
12	99999999	10011001100110011001100110011001	01000110100111100110010011111101
13	aaaaaaaa	10101010101010101010101010101010	01110101101011010101011111001110
14	bbbbbbbb	10111011101110111011101110111011	01100100101111000100011011011111
15	cccccccc	11001100110011001100110011001100	00010011110010110011000110101000
16	dddddddd	11011101110111011101110111011101	00000010110110100010000010111001

No	Plaintext		Ciphertext
	Hexadecimal	Biner	
17	eeeeeeee	1110111011101110111011011101101110	0011000111101001000100111100010 10
18	ffffffff	1111111111111111111111111111111111	0010000011111000000000101001101 1
19	a1a1a1a1	1010000110100001101000011010000 1	0111111010100110010111001100010 1
20	2b2b2b2b	0010101100101011001010110010101 1	1111010000101100110101100100111 1
21	e1e1e1e1	1110000111100001111000011110000 1	0011111011100110000111001000010 1
22	b3b3b3b3	1011001110110011101001110110011	0110110010110100010011101101011 1
23	b4b4b4b4	1011010010110100101101001011010 0	0110101110110011010010011101010 00
24	7c7c7c7c	0111110001111100011111000111110 0	1010001101111011100000010001100 0
25	6a6a6a6a	0110101001101010011010100110101 0	1011010101101101100101110000111 0
26	b6b6b6b6	1011011010110110101101101011011 0	0110100111011000101001011110100 10
27	a3a3a3a3	1010001110100011101000111010001 1	0111110010100100010111101100011 1
28	9d9d9d9d	1001110110011101100111011001110 1	0100001010011010011000001111100 1
29	b5b5b5b5	1011010110110101101101011011010 1	0110101010110010010010001101000 1
30	8a8a8a8a	1000101010001010100010101000101 0	0101010110000110101110111111010 1110

Pada Tabel 6.3 menunjukkan masukkan dari *plaintext* dan hasil dari *ciphertext* yang diperoleh dengan melakukan proses enkripsi. Masukan awal dari *plaintext* yaitu pesan yang merupakan bilangan hexadesimal. Selanjutnya pesan tersebut diubah kedalam bentuk biner agar dapat menghasilkan nilai *ciphertext*.

6.4.2.1 Sniffing Tanpa Enkripsi

Pada bagian ini akan menjelaskan mengenai hasil pengujian dengan melakukan *sniffing* tanpa adanya proses enkripsi yang terjadi pada *slave* sehingga *master* hanya akan menerima *plaintext*.



Gambar 6.2 Screenshoot Hasil Sniffing Tanpa Melakukan Enkripsi

Gambar 6.2 menunjukkan hasil *sniffing* pada serial port yang diperoleh tanpa melakukan proses enkripsi. Pada gambar diatas COM4 berperan sebagai *slave* yang mengirimkan pesan ke *master* dan COM3 berperan sebagai *master* yang menerima pesan dari *slave.* Sedangkan COM6 berperan sebagai *sniffer* yang bertujuan menyadap komunikasi data yang terjadi antara *master* dengan *slave*. Karena tidak dilakukannya proses enkripsi pada *slave*, *sniffer* berhasil menangkap pesan yang sama dengan yang diterima oleh *master* dan pesan yang disadap tersebut merupakan pesan asli.

Tabel 6.4 Hasil Sniffing Tanpa Melakukan Enkripsi

No	Masukan pada Slave	Pesan yang dikirimkan ke Master tanpa enkripsi	Hasil Sniffing yang diperoleh sniffer
1	1010100110101001101 0100110101001	1010100110101001101 0100110101001	1010100110101001101 0100110101001
2	0010111100101111001 0111100101111	0010111100101111001 0111100101111	0010111100101111001 0111100101111
3	0000000000000000000 0000000000000	0000000000000000000 0000000000000	0000000000000000000 0000000000000
4	0001000100010001000 1000100010001	0001000100010001000 1000100010001	0001000100010001000 1000100010001
5	0010001000100010001 0001000100010	0010001000100010001 0001000100010	0010001000100010001 0001000100010

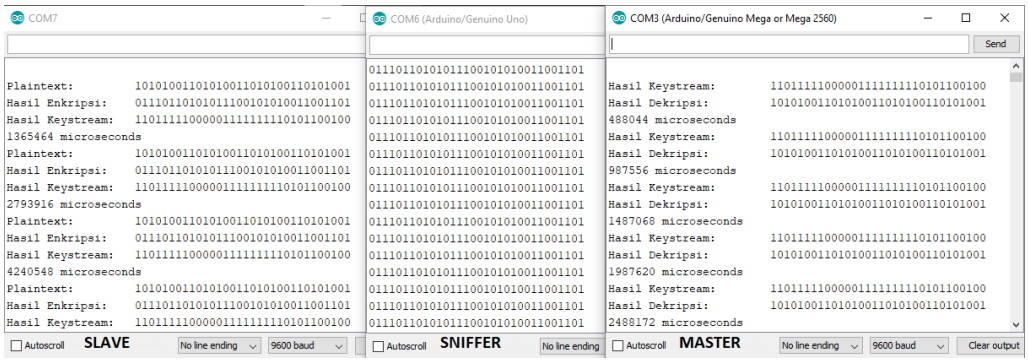
No	Masukan pada Slave	Pesan yang dikirimkan ke Master tanpa enkripsi	Hasil Sniffing yang diperoleh sniffer
6	0011001100110011001 1001100110011	0011001100110011001 1001100110011	0011001100110011001 1001100110011
7	0100010001000100010 0010001000100	0100010001000100010 0010001000100	0100010001000100010 0010001000100
8	0101010101010101010 1010101010101	0101010101010101010 1010101010101	0101010101010101010 1010101010101
9	0110011001100110011 0011001100110	0110011001100110011 0011001100110	0110011001100110011 0011001100110
10	0111011101110111011 1011101110111	0111011101110111011 1011101110111	0111011101110111011 1011101110111
11	1000100010001000100 0100010001000	1000100010001000100 0100010001000	1000100010001000100 0100010001000
12	1001100110011001100 1100110011001	1001100110011001100 1100110011001	1001100110011001100 1100110011001
13	1010101010101010101 0101010101010	1010101010101010101 0101010101010	1010101010101010101 0101010101010
14	1011101110111011101 1101110111011	1011101110111011101 1101110111011	1011101110111011101 1101110111011
15	1100110011001100110 0110011001100	1100110011001100110 0110011001100	1100110011001100110 0110011001100
16	1101110111011101110 1110111011101	1101110111011101110 1110111011101	1101110111011101110 1110111011101
17	1110111011101110111 0111011101110	1110111011101110111 0111011101110	1110111011101110111 0111011101110
18	1111111111111111111 1111111111111	1111111111111111111 1111111111111	1111111111111111111 1111111111111
19	1010000110100001101 0000110100001	1010000110100001101 0000110100001	1010000110100001101 0000110100001
20	0010101100101011001 0101100101011	0010101100101011001 0101100101011	0010101100101011001 0101100101011
21	1110000111100001111 0000111100001	1110000111100001111 0000111100001	1110000111100001111 0000111100001
22	1011001110110011101 1001110110011	1011001110110011101 1001110110011	1011001110110011101 1001110110011

No	Masukan pada Slave	Pesan yang dikirimkan ke Master tanpa enkripsi	Hasil Sniffing yang diperoleh sniffer
23	10110100101101001011010010110100	10110100101101001011010010110100	10110100101101001011010010110100
24	01111100011111001111110001111100	01111100011111001111110001111100	01111100011111000111110001111100
25	01101010011010100110101001101010	01101010011010100110101001101010	01101010011010100110101001101010
26	10110110101101101011011010110110	10110110101101101011011010110110	10110110101101101011011010110110
27	10100011101000110110000111010100	10100011101000110110000111010100	10100011101000111010001110100011
28	10011101100111011001110110011101	10011101100111011001110110011101	10011101100111011001110110011101
29	10110101101101011011010110110101	10110101101101011011010110110101	10110101101101011011010110110101
30	10001010100010101001010100010100	10001010100010101001010100010100	10001010100010101001010100010100

Tabel 6.4 menunjukkan hasil *sniffing* yang diperoleh tanpa melakukan proses enkripsi pada 30 percobaan *plaintext*. Hasilnya menunjukkan bahwa pesan masukan pada *slave* memiliki nilai yang sama dengan pesan yang dikirimkan ke *master* maupun pesan yang disadap oleh *sniffer*.

6.4.2.2 Sniffing Dengan Enkripsi

Pada bagian ini akan menjelaskan mengenai hasil pengujian dengan melakukan *sniffing* dengan melakukan proses enkripsi yang terjadi pada *slave* sehingga *slave* hanya akan mengirimkan pesan yang telah di ekripsi ke *master*.



Gambar 6.3 Screenshoot Hasil Sniffing Dengan Melakukan Enkripsi

Gambar 6.4 menunjukkan hasil *sniffing* yang dilakukan pada ukuran paket data 32 bit. Pada gambar diatas COM7 berperan sebagai *slave* yang melakukan proses enkripsi dan COM3 berperan sebagai *master* yang melakukan proses dekripsi. Sedangkan COM6 merupakan *sniffer* yang bertujuan menyadap komunikasi data yang terjadi antara *master* dengan *slave*. *Sniffer* mampu menyadap pesan yang dikirimkan oleh *master* ke *slave*. Namun, paket data yang disadap oleh *sniffer* bukanlah pesan yang sebenarnya melainkan pesan yang telah di enkripsi oleh *master* sehingga *sniffer* tidak mengetahui pesan yang sebenarnya.

Tabel 6.5 Hasil Sniffing Dengan Melakukan Enkripsi

No	Slave		Hasil enkripsi yang dikirim ke Master	Hasil Sniffing yang diperoleh sniffer
	Masukan Plaintext	Hasil Ciphertext		
1	10101001101010011010100110101001	01110110101011100101010011001101	01110110101011100101010011001101	01110110101011100101010011001101
2	00101111001011110010111100101111	11110000001001000110010010001011	11110000001010001101001001001011	11110000001010001101001001001011
3	00000000000000000000000000000000	11011111000001111111110110001100	11011111000001111111110101100100	11011111000001111111110101100100
4	00010001000100010001000100010001	11001110000101101110110001110101	11001110000101101110110000111 0101	11001110000101101110110000111 0101
5	00100010001000100010001000100010	11111101001001010111011111000110	11111101001001011101111101000110	11111101001001011101111101000110
6	00110011001100110011001100110011	11101100001101001100111001001010111	11101100001101001100111001010111	11101100001101001100111001010111
7	01000100010001000100010001000100	10011011010000111011100100100000	10011011010000111011100100100000	10011011010000111011100100100000
8	01010101010101010101010101010101	10001010010100101010100000110001	10001010010100101010100000110001	10001010010100101010100000110001
9	01100110011001100110011001100110	10111001011000011001101100000010	10111001011000011001101100000010	10111001011000011001101100000010

No	Slave		Hasil enkripsi yang dikirim ke Master	Hasil Sniffing yang diperoleh sniffer
	Masukan Plaintext	Hasil Ciphertext		
10	0111011101110 1110111011101 110111	101010000111 000010001010 00010011	10101000011100 00100010100001 0011	10101000011100 00100010100001 0011
11	1000100010001 0001000100010 001000	010101111000 111101110101 11101100	01010111100011 11011101011110 1100	01010111100011 11011101011110 1100
12	1001100110011 0011001100110 011001	010001101001 111001100100 11111101	01000110100111 10011001001111 1101	01000110100111 10011001001111 1101
13	1010101010101 0101010101010 101010	011101011010 110101010111 11001110	01110101101011 01010101111100 1110	01110101101011 01010101111100 1110
14	1011101110111 0111011101110 111011	011001001011 110001000110 11011111	01100100101111 00010001101101 1111	01100100101111 00010001101101 1111
15	1100110011001 1001100110011 001100	000100111100 101100110001 10101000	00010011110010 11001100011010 1000	00010011110010 11001100011010 1000
16	1101110111011 1011101110111 011101	000000101101 101000100000 10111001	00000010110110 10001000001011 1001	00000010110110 10001000001011 1001
17	1110111011101 1101110111011 101110	001100011110 100100010011 10001010	00110001111010 01000100111000 1010	00110001111010 01000100111000 1010
18	1111111111111 1111111111111 111111	001000001111 100000000010 10011011	00100000111110 00000000101001 1011	00100000111110 00000000101001 1011
19	1010000110100 0011010000110 100001	011111101010 011001011100 11000101	01111110101001 10010111001100 0101	01111110101001 10010111001100 0101
20	0010101100101 0110010101100 101011	111101000010 110011010110 01001111	11110100001011 00110101100100 1111	11110100001011 00110101100100 1111
21	1110000111100 0011110000111 100001	001111101110 011000011100 10000101	00111110111001 10000111001000 0101	00111110111001 10000111001000 0101

No	Slave		Hasil enkripsi yang dikirim ke Master	Hasil Sniffing yang diperoleh sniffer
	Masukan Plaintext	Hasil Ciphertext		
22	1011001110110 0111011001110 110011	011011001011 010001001110 11010111	01101100101101 00010011101101 0111	01101100101101 00010011101101 0111
23	1011010010110 1001011010010 110100	011010111011 001101001001 11010000	01101011101100 11010010011101 0000	01101011101100 11010010011101 0000
24	0111110001111 1000111110001 111100	101000110111 101110000001 00011000	10100011011110 11100000010001 1000	10100011011110 11100000010001 1000
25	0110101001101 0100110101001 101010	101101010110 110110010111 00001110	10110101011011 01100101110000 1110	10110101011011 01100101110000 1110
26	1011011010110 1101011011010 110110	011010011011 000101001011 11010010	01101001101100 01010010111101 0010	01101001101100 01010010111101 0010
27	1010001110100 0111010001110 100011	011111001010 010001011110 11000111	01111100101001 00010111101100 0111	01111100101001 00010111101100 0111
28	1001110110011 1011001110110 011101	010000101001 101001100000 11111001	01000010100110 10011000001111 1001	01000010100110 10011000001111 1001
29	1011010110110 1011011010110 110101	011010101011 001001001000 11010001	01101010101100 10010010001101 0001	01101010101100 10010010001101 0001
30	1000101010001 0101000101010 001010	010101011000 110101110111 11101110	01010101100011 01011101111110 1110	01010101100011 01011101111110 1110

Tabel 6.5 menunjukkan hasil *sniffing* yang diperoleh dengan melakukan proses proses enkripsi pada *slave*. Pada pengujian ini menggunakan 30 percobaan *plaintext*. Hasilnya menunjukkan bahwa pesan masukan pada *slave* memiliki nilai yang berbeda dengan pesan yang dikirimkan ke *master* maupun pesan yang disadap oleh *sniffer*. Hal ini dikarenakan pesan tersebut telah melalui proses enkripsi dan pesan yang dikirimkan ke *master* merupakan pesan yang telah di enkripsi. Selanjutnya pada *master* akan dilakukan proses dekripsi untuk mengembalikan pesan semula. Dalam hal ini *sniffer* berhasil melakukan sniffing,

namun pesan yang disadap merupakan pesan yang telah dienkripsi atau *ciphertext*.

6.5 Pengujian Pemrosesan Enkripsi

Pengujian pemrosesan waktu enkripsi dilakukan untuk memperoleh hasil dan selanjutnya melakukan analisis. Proses pengujian ini bertujuan untuk memperoleh kesimpulan dari hasil pengujian.

6.5.1 Prosedur Pengujian

Dalam melakukan pengujian pemrosesan waktu yang dibutuhkan dalam melakukan proses enkripsi , akan digunakan 8, 16 dan 32 bit data *plaintext*. Perbedaan dari ukuran data ini bertujuan untuk menguji performa dari sistem jika menggunakan ukuran data yang berbeda.

6.5.2 Hasil Pengujian

Pada bagian ini akan menjelaskan mengenai hasil pengujian waktu pemrosesan enkripsi.

6.5.2.1 Masukan 8 Bit

Pada Tabel 6.6 menampilkan waktu pemrosesan yang dibutuhkan untuk melakukan enkripsi sebesar 8 bit.

Tabel 6.6 Hasil Enkripsi 8 bit

Percobaan ke- …	Waktu Enkripsi(milisecond)
1	695
2	697
3	695
4	697
5	695
6	697
7	695
8	697
9	695
10	697
11	695
12	696
13	695
14	696

Percobaan ke- …	Waktu Enkripsi(milisecond)
15	696
16	696
17	695
18	695
19	695
20	696
21	697
22	696
23	696
24	696
25	696
26	696
27	696
28	696
29	695
30	696

Tabel 6.6 menjelaskan mengenai hasil waktu pemrosesan untuk enkripsi data sebesar 8 bit.

6.5.2.2 Masukan 16 bit

Pada Tabel 6.7 menampilkan waktu pemrosesan yang dibutuhkan untuk melakukan enkripsi sebesar 16 bit.

Tabel 6.7 Hasil Enkripsi 16 bit

Percobaan ke- …	Waktu Enkripsi(milisecond)
1	950
2	954
3	950
4	954
5	950
6	954
7	953

Percobaan ke- …	Waktu Enkripsi(milisecond)
8	950
9	950
10	950
11	950
12	950
13	950
14	950
15	953
16	953
17	953
18	950
19	950
20	951
21	950
22	950
23	950
24	950
25	950
26	950
27	953
28	953
29	953
30	953

Tabel 6.7 menjelaskan mengenai hasil waktu pemrosesan untuk enkripsi data sebesar 16 bit.

6.5.2.3 Masukan 32 bit

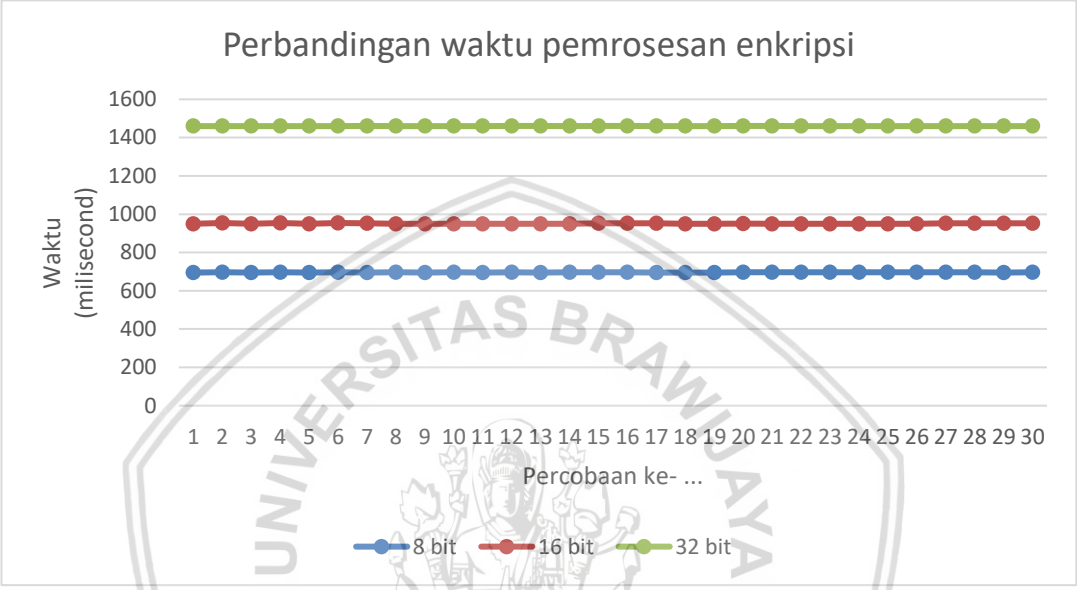
Pada Tabel 6.8 menampilkan waktu pemrosesan yang dibutuhkan untuk melakukan enkripsi sebesar 32 bit.

Tabel 6.8 Hasil Enkripsi 32 bit

Percobaan ke- …	Waktu Enkripsi(milisecond)
1	1460
2	1460
3	1460
4	1460
5	1460
6	1460
7	1461
8	1460
9	1460
10	1460
11	1460
12	1461
13	1461
14	1460
15	1461
16	1461
17	1460
18	1460
19	1460
20	1461
21	1461
22	1460
23	1460
24	1460
25	1460
26	1460
27	1460
28	1460
29	1460

Percobaan ke- …	Waktu Enkripsi(milisecond)
30	1460

Tabel 6.8 menjelaskan mengenai hasil waktu pemrosesan untuk enkripsi data sebesar 32 bit.



Gambar 6.4 Grafik proses enkripsi

Pada gambar 6.4 menunjukkan perbedaan hasil waktu yang dibutuhkan untuk melakukan proses enkripsi dari masing-masing ukuran data 8 bit, 16 bit dan 32 bit. Dari grafik diatas dapat disimpulkan bahwa enkripsi *plaintext* 32 membutuhkan waktu pemrosesan enkripsi tertinggi dibandingkan dengan *plaintext* 8 bit dan 16 bit.

One-Sample Statistics

	N	Mean	Std. Deviation	Std. Error Mean
8 bit	30	695.8333	.74664	.13632
16 bit	30	951.2333	1.61210	.29433
32 bit	30	1460.2333	.43018	.07854

Gambar 6.5 Hasil Perhitungan Rata-rata Waktu Pemrosesan Enkripsi

Pada Gambar 6.5 menjelaskan hasil perhitungan rata-rata waktu pemrosesan enkripsi yang diperoleh dari masing-masing ukuran data 8,16 dan 32 bit. Hasil rata-rata waktu pemrosesan enkripsi yang diperoleh oleh ukuran data 8 bit yaitu sebesar 695,8 ms. Sedangkan hasil rata-rata waktu pemrosesan enkripsi yang diperoleh oleh ukuran data 16 bit yaitu sebesar 951,2 ms. Dan hasil rata-rata waktu pemrosesan enkripsi yang diperoleh oleh ukuran data 32 bit yaitu sebesar 1460,2 ms.

6.6 Pengujian Pemrosesan Dekripsi

Pengujian pemrosesan waktu dekripsi dilakukan untuk memperoleh hasil dan selanjutnya melakukan analisis. Proses pengujian ini bertujuan untuk memperoleh kesimpulan dari hasil pengujian.

6.6.1 Prosedur Pengujian

Dalam melakukan pengujian pemrosesan waktu yang dibutuhkan, akan digunakan 8, 16 dan 32 bit data *plaintext*. Perbedaan dari ukuran data ini bertujuan untuk menguji performa dari sistem jika menggunakan ukuran data yang berbeda.

6.6.2 Hasil Pengujian

Pada bagian ini akan menjelaskan mengenai hasil pengujian waktu pemrosesan dekripsi.

6.6.2.1 Hasil Dekripsi 8 bit

Pada Tabel 6.9 menampilkan waktu pemrosesan yang dibutuhkan untuk melakukan dekripsi paket data ukuran 8 bit.

Tabel 6.9 Hasil Dekripsi 8 bit

Percobaan ke- …	Waktu Dekripsi (milisecond)
1	444
2	444
3	444
4	444
5	444
6	444
7	443
8	443
9	443
10	443
11	443
12	444
13	444
14	444
15	443
16	444

Percobaan ke- …	Waktu Dekripsi (milisecond)
17	444
18	443
19	443
20	443
21	442
22	444
23	443
24	444
25	444
26	444
27	444
28	444
29	443
30	443

Tabel 6.9 menjelaskan mengenai hasil waktu pemrosesan untuk dekripsi data sebesar 8 bit.

6.6.2.2 Hasil Dekripsi 16 bit

Pada Tabel 6.10 menampilkan waktu pemrosesan yang dibutuhkan untuk melakukan dekripsi paket data ukuran 16 bit.

Tabel 6.10 Hasil Dekripsi 16 bit

Percobaan ke- …	Waktu Dekripsi (milisecond)
1	448
2	448
3	447
4	448
5	448
6	447
7	448
8	448
9	448

Percobaan ke- …	Waktu Dekripsi (milisecond)
10	448
11	448
12	447
13	447
14	448
15	447
16	448
17	448
18	448
19	448
20	448
21	448
22	448
23	448
24	448
25	448
26	448
27	448
28	448
29	448
30	448

Tabel 6.10 menjelaskan mengenai hasil waktu pemrosesan untuk dekripsi data sebesar 16 bit.

6.6.2.3 Hasil Dekripsi 32 bit

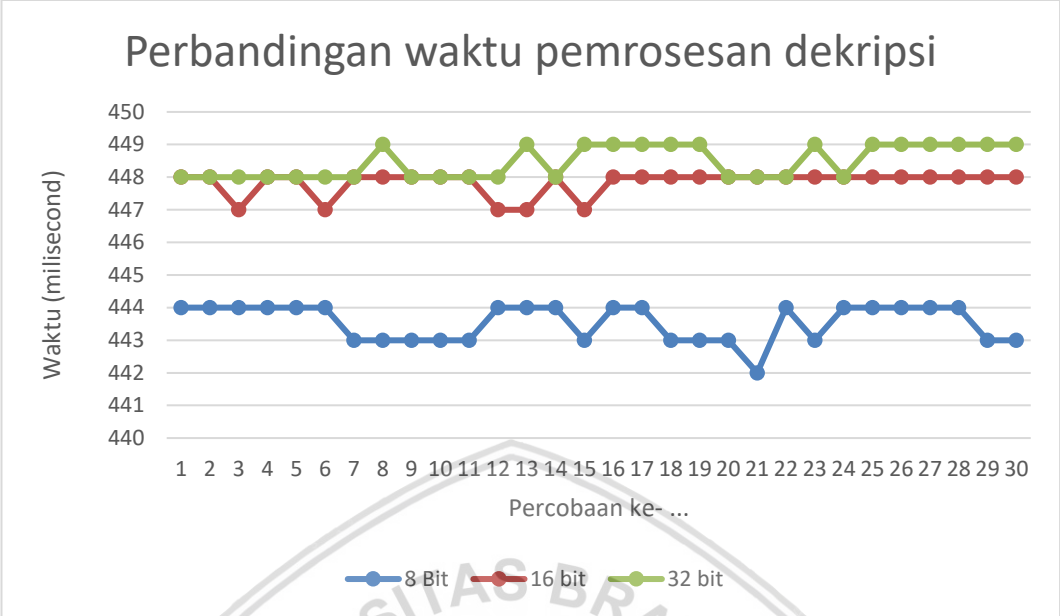
Pada Tabel 6.11 menampilkan waktu pemrosesan yang dibutuhkan untuk melakukan dekripsi paket data ukuran 32 bit.

Tabel 6.11 Hasil Dekripsi 32 bit

Percobaan ke- …	Waktu Dekripsi (milisecond)
1	448
2	448

Percobaan ke- …	Waktu Dekripsi (milisecond)
3	448
4	448
5	448
6	448
7	448
8	449
9	448
10	448
11	448
12	448
13	449
14	448
15	449
16	449
17	449
18	449
19	449
20	448
21	448
22	448
23	449
24	448
25	449
26	449
27	449
28	449
29	449
30	449

Tabel 6.11 menjelaskan mengenai hasil waktu pemrosesan untuk dekripsi data sebesar 32 bit.



Gambar 6.6 Grafik Proses Dekripsi

Pada gambar 6.6 menunjukkan perbedaan hasil waktu yang dibutuhkan untuk melakukan proses dekripsi dari masing-masing ukuran data 8 bit, 16 bit dan 32 bit. Pada grafik diatas dapat disimpulkan bahwa proses dekripsi plaintext 32 bit membutuhkan waktu pemrosesan yang lebih tinggi dibandingkan dengan *plaintext* berukuran 8 bit dan 16 bit.

One-Sample Statistics

	N	Mean	Std. Deviation	Std. Error Mean
8 bit	30	443.5333	.57135	.10431
16 bit	30	447.8333	.37905	.06920
32 bit	30	448.4667	.50742	.09264

Gambar 6.7 Hasil Perhitungan Rata-rata Waktu Pemrosesan Dekripsi

Pada Gambar 6.7 menjelaskan hasil perhitungan rata-rata waktu pemrosesan dekripsi yang diperoleh dari masing-masing ukuran data 8,16 dan 32 bit. Hasil rata-rata waktu pemrosesan dekripsi yang diperoleh oleh ukuran data 8 bit yaitu sebesar 443,5 ms. Sedangkan hasil rata-rata waktu pemrosesan dekripsi yang diperoleh oleh ukuran data 16 bit yaitu sebesar 447,8 ms. Dan hasil rata-rata waktu pemrosesan dekripsi yang diperoleh oleh ukuran data 32 bit yaitu sebesar 448,4 ms.

6.7 Pengujian Waktu Pengiriman Hasil Enkripsi Dari Master ke Slave

Pengujian waktu pengiriman hasil enkripsi dari *master* ke *slave* dilakukan untuk memperoleh hasil dan selanjutnya melakukan analisis. Pengujian ini bertujuan untuk memperoleh kesimpulan dari hasil pengujian.

6.7.1 Prosedur Pengujian

Dalam melakukan pengujian pemrosesan waktu pengiriman yang dibutuhkan, akan digunakan 8, 16 dan 32 bit data *plaintext*. Perbedaan dari ukuran data ini bertujuan untuk menguji performa dari sistem dalam mengirimkan paket data dengan menggunan ukuran data yang berbeda.

6.7.2 Hasil Pengujian

Pada bagian ini akan menjelaskan mengenai hasil pengujian waktu pengiriman hasil enkripsi yang dilakukan oleh *slave*.

6.7.2.1 Masukan 8 bit

Pada Tabel 6.12 menampilkan waktu yang dibutuhkan untuk melakukan pengiriman hasil enkripsi data dengan ukuran 8 bit.

Tabel 6.12 Hasil Pengiriman 8 bit

Percobaan ke- …	Waktu(microsecond)
1	12
2	12
3	11
4	12
5	12
6	12
7	12
8	12
9	12
10	12
11	12
12	11
13	12
14	12
15	12
16	11
17	12
18	12
19	12
20	11

Percobaan ke- …	Waktu(microsecond)
21	12
22	12
23	11
24	11
25	11
26	12
27	11
28	12
29	11
30	12

Tabel 6.12 menjelaskan mengenai hasil waktu waktu yang dibutuhkan untuk melakukan pengiriman hasil enkripsi data dengan ukuran 8 bit.

6.7.2.2 Masukan 16 bit

Pada Tabel 6.13 menampilkan waktu yang dibutuhkan untuk melakukan pengiriman hasil enkripsi data dengan ukuran 16 bit.

Tabel 6.13 Hasil Pengiriman 16 bit

Percobaan ke- …	Waktu(milisecond)
1	13
2	13
3	13
4	13
5	13
6	13
7	13
8	13
9	13
10	13
11	13
12	13
13	13

Percobaan ke- …	Waktu(milisecond)
14	13
15	13
16	12
17	13
18	12
19	12
20	12
21	12
22	12
23	13
24	13
25	13
26	13
27	13
28	13
29	12
30	13

Tabel 6.13 menjelaskan mengenai hasil waktu waktu yang dibutuhkan untuk melakukan pengiriman hasil enkripsi data dengan ukuran 16 bit.

6.7.2.3 Masukan 32 bit

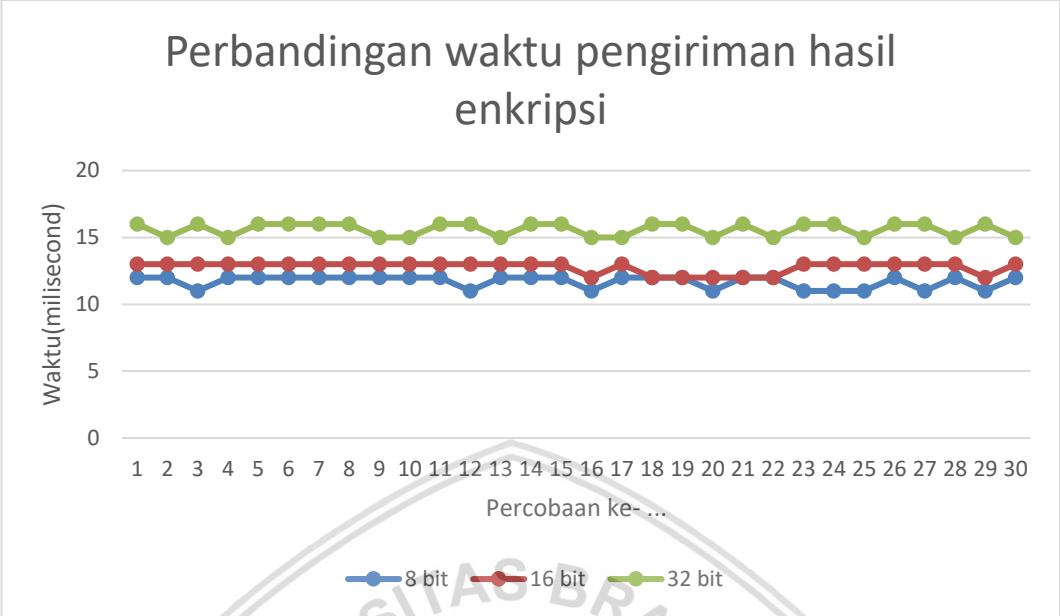
Pada Tabel 6.14 menampilkan waktu pemrosesan yang dibutuhkan untuk melakukan pengiriman hasil enkripsi data dengan ukuran 32 bit.

Tabel 6.14 Hasil Pengiriman 32 bit

Percobaan ke- …	Waktu(milisecond)
1	16
2	15
3	16
4	15
5	16
6	16

Percobaan ke- …	Waktu(milisecond)
7	16
8	16
9	15
10	15
11	16
12	16
13	15
14	16
15	16
16	15
17	15
18	16
19	16
20	15
21	16
22	15
23	16
24	16
25	15
26	16
27	16
28	15
29	16
30	15

Tabel 6.14 menjelaskan mengenai hasil waktu yang dibutuhkan untuk melakukan pengiriman hasil enkripsi data dengan ukuran 32 bit.



Gambar 6.8 Grafik Waktu Pengiriman Hasil Enkripsi

Pada gambar 6.8 menunjukkan perbedaan hasil waktu yang dibutuhkan untuk melakukan pengiriman hasil enkripsi data 8 bit, 16 bit dan 32 bit. Dari grafik diatas dapat disimpulkan bahwa waktu pengiriman hasil enkripsi tertinggi diperoleh oleh ukuran data 32 bit.

One-Sample Statistics

	N	Mean	Std. Deviation	Std. Error Mean
8 bit	30	11.7000	.46609	.08510
16 bit	30	12.7667	.43018	.07854
32 bit	30	15.6000	.49827	.09097

Gambar 6.9 Hasil Perhitungan Rata-rata Waktu Pengiriman Hasil Enkripsi Dari *Slave*

Pada Gambar 6.9 menjelaskan hasil perhitungan rata-rata waktu pengiriman hasil enkripsi dari slave yang diperoleh dari masing-masing ukuran data 8,16 dan 32 bit. Hasil rata-rata waktu pengiriman hasil enkripsi yang diperoleh oleh ukuran data 8 bit yaitu sebesar 11,7 ms. Sedangkan hasil rata-rata waktu pengiriman hasil enkripsi yang diperoleh oleh ukuran data 16 bit yaitu sebesar 12,7 ms. Dan hasil rata-rata waktu pengiriman hasil enkripsi yang diperoleh oleh ukuran data 32 bit yaitu sebesar 15,6 ms.

BAB 7 PENUTUP

7.1 Kesimpulan

Kesimpulan yang dapat diambil dari penelitian implementasi algoritme Trivium untuk mengamankan komunikasi data *master-slave* pada perangkat berbasis modul komunikasi MRF24L01 ini adalah sebagai berikut:

1. Hasil validasi *keystream* yang dikeluarkan oleh sistem menunjukkan hasil yang sama dengan *test vector*.
2. Kinerja dari pengenkripsian dengan menggunakan modul komunikasi NRF24L01 menunjukkan bahwa algoritme Trivium dapat berkerja dengan baik melalui pengujian *sniffing* dimana pesan asli tidak dapat disadap oleh *sniffer*.
3. Kinerja pemrosesan enkripsi algoritme Trivium dengan menggunakan perangkat berbasis modul komunikasi NRF24L01 memiliki rata-rata 695,8 ms untuk ukuran data 8 bit, 951,2 ms untuk ukuran data 16 bit dan 1460,2 ms untuk ukuran data 32 bit.
4. Kinerja pemrosesan dekripsi algoritme Trivium dengan menggunakan perangkat berbasis modul komunikasi NRF24L01 memiliki rata-rata 443,5 ms untuk ukuran data 8 bit, 447,8 ms untuk ukuran data 16 bit dan 448,4 ms untuk ukuran data 32 bit.

7.2 Saran

Saran yang dapat diberikan untuk penelitian selanjutnya yang akan mengembangkan sistem ini antara lain:

1. Proses penentuan *keystream* untuk penelitian selanjutnya pada key dan IV dapat dilakukan secara manual dengan masukan yang sesuai dengan keinginan pengguna.
2. Pada penelitian ini masukan enkripsi dan dekripsi masih menggunakan sampel acak diharapkan untuk pengembangan selanjutnya dapat menggunakan data yang valid.

DAFTAR PUSTAKA

Ali, Z., Rahim, A. and Nawaz, S.S., 2016. Design and Implementation of a Low Cost Wireless Sensor Network using Arduino and NRF24L01 (+). In *Second National Conference on Emerging Trends in Computer Science & Engineering (NCETCSE-2016), 29th April*.

Anon., 2018. Arduino MEGA 2560 & Genuino MEGA 2560. [online] Available at: https://www.arduino.cc/en/Main/ArduinoBoardMegaADK?from=Main.ArduinoBoardADK [Diakses 18 july 2018].

Anon.,2018.Confidental Information Protection. [online] Available at: https://infowatch.com/solutions/information_security [Diakses 11 Maret 2018].

Boruchinkin, A., Tolstaya, A. and Zhgilev, A., 2018. Cryptographic Wireless Communication Device. *Procedia Computer Science*, *123*, pp.110-115.

Christ, P., Neuwinger, B., Werner, F. and Rückert, U., 2011, October. Performance analysis of the NRF24L01 ultra-low-power transceiver in a multi-transmitter and multi-receiver scenario. In *Sensors, 2011 IEEE* (pp. 1205-1208). IEEE.

De Canniere, C., 2006, August. Trivium: A stream cipher construction inspired by block cipher design principles. In *International Conference on Information Security* (pp. 171-186). Springer, Berlin, Heidelberg.

Jafarpour, A., Mahdlo, A., Akbari, A. and Kianfar, K., 2011, December. Grain and Trivium ciphers implementation algorithm in FPGA chip and AVR micro controller. In *Computer Applications and Industrial Electronics (ICCAIE), 2011 IEEE International Conference on* (pp. 656-658). IEEE.

Marmolejo-Tejada, J.M., Trujillo-Olaya, V. and Velasco-Medina, J., 2010, September. Hardware implementation of grain-128, mickey-128, decim-128 and Trivium. In *ANDESCON, 2010 IEEE* (pp. 1-6). IEEE.

Ken, C. and Xiaoying, L., 2012. A SOPC-BASED Evaluation of AES for 2.4 GHz Wireless Network. *Physics Procedia*, *33*, pp.1481-1488.

Kitsos, P., Sklavos, N., Provelengios, G. and Skodras, A.N., 2013. FPGA-based performance analysis of stream ciphers ZUC, Snow3g, Grain V1, Mickey V2, Trivium and E0. *Microprocessors and Microsystems*, *37*(2), pp.235-245.

Zarqoni, M., 2017. *Perancangan Dan Implementasi Pemilihan Cluster Head Dan Koordinasi Antar Node Sensor Pada Jaringan Sensor Nirkabel Dengan Modul Wireless NRF24L01*(Doctoral dissertation, Institut Teknologi Sepuluh Nopember).